Conception d'un Modèle de Prédiction des Défauts de Paiement sur les Prêts : Approches, Algorithmes et Évaluation comparative

Yacouba Diarra

Juin 2024

Résumé

La démocratisation des prêts bancaire fut un pilier fondamental pour l'économie moderne. De par leur prêts, les banques et autres institutions financières ont pu aider des centaines de millions de personnes à s'offrir une résidence, payer des frais médicaux, un enseignement supérieur, créer une entreprise et j'en passe. Différents systèmes de prêts ont d'ailleurs depuis longtemps supporter une grande partie de l'économie dans diverses civilisations au cours de l'histoire mais il est clair qu'ils n'ont jamais été aussi fondamentaux au point d'inspirer la création de modèles économiques entiers autour d'eux.

Ceci étant dit, ils comportent un risque majeur que les institutions financières et juridiques ont toujours cherché à minimiser : *les défauts de paiement*. La décision d'accorder ou non un prêt à tel individu ou tel organisation est une tâche délicate qui nécessite beaucoup d'attention, car s'il est vrai qu'un prêt peut changer la vie de quelqu'un, des mauvaises décisions répétées représente un risque sérieux pour la stabilité de l'institution prêteuse et un fardeau pésant pour le client.

Dans ce mémoire, je propose une étude approfondie des dernières méthodes employées par ces institutions pour attester la capacité de remboursement des clients, visant à prédire quels clients présentent le plus de risque de faire défaut sur un prêt grâce à l'apprentissage automatique.

Mots-clés : Défaut de payement, Science des données, Apprentissage automatique

Remerciements

Ça fait probablement un peu cliché de dire ça mais bien évidemment les premières personnes que je tiens à remercier dans ce document ce sont mes parents, ça a commencé au jardin d'enfants et aujourd'hui c'est pour un diplôme de licence donc merci. Ensuite je tiens à remercier Robots Mali pour un stage qui, sans mentir, a changé ma vie et considérablement élargi mes horizons. Un remerciement spécial aussi pour mon directeur de mémoire, Dr Maïga Oumar pour sa contribution aux recherches et sa supervision. Aussi à mes camarades de classe pour les expériences qu'on a partagé durant presque 3 ans maintenant, je n'ai aucun doute que ces expériences ont aussi servi à rédiger ce mémoire.

Merci à Kaggle et à Home Credit Group pour l'environnement de travail et les données que j'ai utilisé pour mes recherches et plus généralement pour leur contibution à la communauté Machine learning et IA.

Enfin merci à toutes les personnes qui ont pris part à la relecture du document pour leur retours et leur aide pour améliorer la qualité et la clarté de ce mémoire.

Table des matières

Avant-propos

Le document comporte six (6) chapitres repartis en trois grande partie. La première partie (”Introduction”) est pensée pour apporter les connaissances de base des différents domaines d'expertise impliqués dans cette étude ainsi que pour introduire de manière plus formelle le thème, ses spécificités et ses implications. Cette première partie contient les deux premiers chapitres portant sur les deux piliers de ce thème, respectivement le système de prêt et la science des données. L'objectif étant d'introduire les concepts fondamentaux qui englobent les différentes techniques que j'ai employé dans mes recherches.

La deuxième partie rentre dans les détails de cette méthodologie, cette partie se veut aussi détaillée que possible afin d'apporter une compréhension complète du travail éffectué. En partant des concepts généraux jusqu'aux utilisations et implémentations spécifiques que j'en ai fait j'espère démystifier le processus avant de présenter les résultats obtenu. Le chapitre 3 concerne exclusivement les manipulations et traitements appliqués aux données qui ont servi à dévélopper les différents modèles tandis que le chapitre 4 explique le fonctionnement et l'implémentation de ces modèles. Enfin le chapitre 5 détaillera le processus d'évaluation des différents modèles entraînés avant de procéder à une évaluation comparatives des différents algorithmes.

Finalement la partie conclusion et le dernier chapitre visent à offir un résumé concis du document, à apporter des réponses à certaines questions que vous vous poserez probablement après votre lecture, à mettre en lumière les challenges rencontrés au cours des recherches et enfin à souligner des pistes de recherches potentielles pour de futures avancées concernant les systèmes de prédiction de défaut de paiement.

Je tenais à faire ce mémoire le plus explicite possible afin qu'aucune connaissance préalable ne soit nécéssaire pour comprendre les différents concepts. Toutefois, quelques connaissances de bases en statisques, theorie de la probabilité, algèbre linéaire et algorithmique à un niveau sophomore pourraient grandement faciliter la compréhension de certains concepts.

Première partie

Introduction

Chapitre 1

Prédire les défauts de paiement

Dans ce premier chapitre je cherche à formaliser le thème et à placer le contexte dans lequel il est pertinent. Celà implique de s'interesser au monde de la finance et de chercher à comprendre les fondements de l'économie mondiale dans le passé et aujourd'hui afin de réellement prendre conscience de la place qu'occupent les prêts dans celle ci. Nous discuterons également brièvement les différentes approches que les institutions financières utilisent pour évaluer les demandes de prêt. Vous l'aurez compris ce premier chapitre s'interesse au pourquoi et au comment de notre thème.

1.1 Contexte

Dans notre société contemporaine, l'économie à l'échelle mondiale se répose de plus en plus sur les prêts. Cette situation est notamment favorisée par le fait qu'aujourd'hui toutes les économies se basent sur une forme de capitalisme [1], les économies socialistes [2] étant souvent taxée comme restrictives de liberté et empêchant le dévéloppement économique des individus au profit de celui de l'état. Cette mentalité inspirée surtout par ***"The american dream"***, le rêve américain en français, a pas mal chambouler les normes, maintenant on a tous un peu l'idée que pour être quelqu'un il faut avoir une villa, une voiture et des bouts de terre à son nom.

Ce changement de mentalité a été le point de départ d'importants changements dans l'économie mondiale. Le rêve américain s'est démocratisé et est de-

1. Le capitalisme est un système économique caractérisé par la propriété privée des moyens de production, des marchés concurrentiels et la recherche du profit.

2. Le socialisme est un système politique et économique dans lequel la propriété et les moyens de production sont détenus en commun, généralement contrôlés par l'État ou le gouvernement. Le socialisme repose sur l'idée que la propriété commune ou publique des ressources et des moyens de production conduit à une société plus égalitaire.

venu un rêve mondial, apportant l'idée que chaque personne a la liberté et la possibilité de réussir et d'accéder à une vie meilleure. C'est donc à partir de là que le système de prêt est devenu réellement LE pilier de l'économie mondiale avec un impact absolument démésuré. D'après la banque mondiale, le crédit mondial au secteur privé représentait environ 97,6% du PIB mondial en 2020 (BANK, 2024).

Les prêts ont definitivement changé de statut et l'économie mondiale de visage, les individus emprûntent de l'argent pour s'acheter une maison, une boutique, une voiture, en bref vivre le rêve américain mais aussi pour faire des études supérieures. L'americain moyen passe typiquement les premières années de sa vie professionelle à rembourser son prêt étudiant. Au delà des individus les gouvernements contribuent à renforcer ce système souvent à travers des rélations complexes avec les banques.

Mais là où l'impact est le plus grand est sans aucun doute le secteur de l'entreprenariat privé. Il est vraiment impressionant de se dire qu'il y a des entreprises aujourd'hui évaluées à des centaines de millions voir des milliards de dollars qui partent vraiment juste d'une bonne idée et d'un investissement. Pendant la crise du coronavirus, les crédits ont été le bienfaiteur qui a sauvé des milliers de petites et moyennes entreprises à travers le monde et paradoxalement ils ont aussi été la raison qui a précipité la chutte de centaines d'entre elles.

Les dettes, c'est souvent le commencement de la ruine. Après la crise du covid-19, le FC Barcelone (incontestablement un des plus grands clubs de football de l'histoire) annoncait une dette globale de plus de 2 milliards d'euros, ce fut le début d'une période très difficile que le club traverse encore aujourd'hui. Pas seulement dans le domaine du sport, tellement d'entreprises et organisation ont dû déposer le bilan à cause de leur dettes, les prêts font les grands et les dettes les détruisent. On ne crée pas de richesse à partir de rien, ce qui fait que même les banques et les institutions prêteuse sont en réalité souvent très endettées. Pour rappel, la banque d'investissement des frères Lehman a déposé son bilan en raison de son exposition massive aux dettes hypothécaires à risque, ce qui a conduit à la grande crise financière de 2008.

C'est probablement après cet évènement que les gens et en particulier les américains ont réellement réalisé que l'énorme croissance économique que l'on a connu après la seconde guerre mondiale s'est faite sur des dettes cumulées énormes des grandes institutions et qu'on ferait mieux ne pas jouer avec ça. Suivant la crise plusieurs lois ont été votées ou modifiées un peu partout dans le monde afin d'eviter un tel désastre dans le futur, mettant l'accent sur la protection du client mais aussi la transparence des procedures.

Vous pensez peut-être que toutes ces choses sont des problèmes de la société contemporaine mais détrompez-vous, j'ai été surpris d'apprendre que bien qu'ils n'avaient jamais atteint un tel niveau d'importance auparavant différents systèmes de prêt ont en réalité soutenu les économies de plusieurs civilisations par

le passé. Bien que ces concepts et lois ont évolué avec le temps, ils sont enfaîte extrêmement anciens. Autour de moins 2000 ans avant Jésus Christ en Mésopotamie, les temples et les palais prêtaient du grain aux agriculteurs et aux commerçants avec des formes de contrat incluant des taux d'interêt, des clauses et des délais, un peu comme les prêts d'aujourd'hui. Le **code d'Hammurabi**, un texte légal Babylonien composé entre 1755 et 1750 avant Jésus Christ incluait déjà des règles sur les prêts et les taux d'intérêt. Il s'agit du texte juridique le plus long, le mieux organisé et le mieux conservé de l'ancienne Mésopotamie. Il est écrit dans le vieux dialecte babylonien de l'akkadien, prétendument par Hammurabi, sixième roi de la première dynastie de Babylone.

Ça fait beaucoup de choses qu'un informaticien de la génération Z n'est pas vraiment censé savoir mais s'il y'a une chose que j'ai bien compris c'est qu'avec le model économique actuelle le système de prêt est à proteger absolument au risque de l'éffondrement de l'économie mondiale. Dans cette optique, une décision en particulier devient plus crucial que le reste : À qui accorder un prêt ? Pour proteger à la fois le client et le prêteur, il est primordial de pouvoir déterminer avec précision quel client est ou sera en capacité de rembourser tel prêt. Voilà qui nous amène à notre thème maintenant, dans la prochaine section nous allons discuter le comment maintenant.

1.2 Approches

Les banques sont tenues de respecter les normes réglementaires qui imposent des pratiques de prêt prudentes. Pouvoir prédire de manière précise qui peut ou ne peut pas rembourser tel prêt aident à respecter ces normes et à éviter les pénalités.

Pour prédire les défauts de paiement, les banques analysaient traditionnellement divers facteurs, parmi lesquels :

- Historique de crédit : un enregistrement détaillé du comportement d'emprunt et de remboursement passé du demandeur.
- Revenu et emploi : un revenu constant et suffisant indique une capacité à rembourser le prêt.
- Ratio dette/revenu : il mesure le fardeau de la dette du demandeur par rapport à son revenu.
- Valeur de la garantie : pour les prêts garantis, la valeur de la garantie est cruciale en cas de défaut de paiement de l'emprunteur.

Typiquement une demande de prêt est analysée méticuleusement suivant ces différents aspects par des professionels du domaine qui en fonction de ces élément decident d'accorder ou de refuser le prêt au demandeur. Vous l'aurez compris, les antécédents priment beaucoup dans la procedure, un jeune diplômé qui

vient d'obtenir son premier contrat à durée indéterminée (CDI) n'a presque aucune chance de pouvoir obtenir un prêt, à moins d'avoir une garantie de fer.

Des méthodes un peu plus modernes peuvent également inclure des données comportementales, telles que les habitudes de dépenses et même l'activité sur les réseaux sociaux. Oui ça commence à faire un peu beaucoup, ce qui nous amène encore une fois directement au thème de ce mémoire. L'étude d'une demande de prêt est une procédure délicate qui augmente en complexité avec différents facteurs. Les données à analyser aussi croissent en complexité et surtout en quantité multipliant le risque d'errreur humaine dans la procédure.

Des mauvaise prévisions repétées peuvent avoir de graves répercussions :

— Augmentation des pertes sur prêts : un nombre plus élevé de défauts de paiement a un impact direct sur les résultats financiers de la banque.
— Taux d'intérêt plus élevés : pour couvrir les pertes potentielles, les banques pourraient augmenter les taux d'intérêt pour tous les emprunteurs, rendant les prêts plus chers.
— Resserrement du crédit : les banques pourraient devenir plus conservatrices dans leurs prêts, ce qui rendrait plus difficile l'obtention de prêts pour les particuliers et les entreprises.
— Dommages à la réputation : des taux de défaut élevés peuvent nuire à la réputation d'une banque, affectant la confiance des clients et ses activités futures.

L'apprentissage automatique ou apprentissage machine, un sous-domaine de l'intelligence artificiel centré sur la création d'algorithme qui puisse extraire, interprêter et généraliser des connaissances à partir des données, a apporté de nouvelles méthodes visant à assister cette procédure par les machines, améliorant considérablement la précision et l'efficacité. Ces nouvelles méthodes n'ont pas vocation de remplacer les professionnels du domaine mais plutôt de les épauler avec des données qui deviennent de plus en plus large et à garder les prêts accessible à ceux qui le méritent le plus. Ces algorithmes sont capables de repérer des tendances dans les données, des corrélations entres différents paramètres et les comportements et decision des clients, créer un modèle statisque de ce qui représente un bon prêt et en fin de compte prédire les chances qu'un client a de faire défaut sur un prêt donné. Les machines ont un clair avantages sur nous en terme de puissance et de vitesse de calcul, les nouveaux outils apportés par l'apprentissage automatique en particulier et la science des données en général se sont donc vite imposés dans le flux de travail des banques et autres institutions financières avec un potentiel énorme pour rendre les prêts plus accessibles et plus sûrs.

Imaginez devoir analyser manuellement des centaines voir des milliers de demande en fonction d'une quantité colossale de variables, en plus des experts en finances et en économie qu'on suppose qu'ils ont déjà, une banque aura besoin

de statisticiens, probablement de juristes et de gens qui connaissent la réalité des différents domaines proposé par les démandeurs, des *"experts de domaine"*. Par exemple, il y'a encore 30 ans beaucoup ne donnait pas cher de la peau de l'internet et des nouvelles compagnie qui se basaient sur lui. Quand on voit ce qu'est devenu Facebook aujourd'hui, très mauvais pari effectivement. De manière similaire, même des experts de la finance se moquaient du **Bitcoin** au début des années 2010 et ne cessaient de prédire sa chutte imminente. L'économie est une chose complexe où des tendances se créent et s'inversent à chaque décennie, prendre des décisions informées implique de se mettre constamment à jour avec ce monde.

La promesse de ces nouvelles techniques est de pouvoir comprendre et même prédire ces tendances de manière mathématiques. Dans ce mémoire, nous allons procéder à une étude approfondie de ces nouvelles techniques provenant de la science des données et de l'apprentissage automatique couvrant aussi bien leur fonctionnement que les algorithmes qui sont derrière, ainsi que leur application dans le monde réel. La dernière section de ce premier chapitre apporte plus de détails sur les spécifications actuelles du travail éffectué.

1.3 Introduction formelle du thème

Bien que ces nouvelles approches ne sont pas encore largement répendues et utilisées par les banques au Mali, elles sont beaucoup plus communes en occident et déjà adoptées par les grandes institutions. Donc des progrès déjà significatifs ont été réalisés dans le domaine, enfaîte la prediction des défauts de paiement est l'une des plus vielles application de l'apprentissage automatique, un classique.

Ce que je propose avec ce thème est donc une analyse de cet état de l'art et du flux de travail associé, pour celà j'ai implémenté certaines des dernières techniques de prétraitement et d'analyse des données ainsi que les modèles d'apprentissage automatiques qui ont atteint l'état de l'art dans cette tâche. J'ai voulu diverger de la majorité des chercheurs en testant aussi des algorithmes moins conventionnels et moins populaires pour la prediction de défaut de paiement, qui m'ont, au final, agréablement surpris par leur résultats au moment des tests.

Au total cinq (5) modèles d'apprentissage automatique différents ont été entraînés et testés, des valeurs sûres comme la régression logistique, la forêt aléatoire et le boost de gradient mais aussi les réseaux de neuronnes artificiels (souvent mis de côté pour ce genre de tâche en raison de leur complexité) et la détection d'anomalie qui à la base n'est pas un algorithme de classification mais qui à démontré la meilleure précision avec des données déséquilibrées. Le chapitre 4 va en profondeur dans les processus d'entraînement et les différences conceptuelles de chacun de ces modèles ainsi que les implémentations et stratégies que

j'ai utilisé.

Ces cinq modèles ont tous été entraînés et testés suivant le même canevas ou *pipeline* et avec le même ensemble de données, afin que les performances ne puisse différer que de part les caractéristiques intrinsèques de chaque modèle limitant leur performances après un certains nombre d'étapes d'optimisation ainsi évitant de biaser leur évaluation dans un système ou un modèle est avantagé par rapport aux autres.

L'ensemble de données en question est constituée d'exactement 100 variables indépendantes ou prédicateurs (**features**) et d'une variable dépendante (celle qu'on cherche à prédire, **target**), à savoir si oui ou non le client en question a remboursé son prêt. Totalisant 1.526.659 exemples étiquétés qui seront divisés en deux groupes pour l'entraînement et le test. Ces données ont été obtenu après une étape d'ingénierie de prédicateurs ou création de prédicateurs et une étape de sélection et de prétraitement des variables indépendantes (détaillées dans le chapitre 3) que j'ai appliqué aux données brutes fournies par Home Credit Group, une institution financière non bancaire internationale fondée en 1997 en République tchèque et dont le siège est aux Pays-Bas, lors d'une competition qu'ils organisaient sur la plateforme mondiale des scientifiques des données et d'ingénieurs en apprentissage automatique, **Kaggle** (Herman et al., 2024). La société opère dans 9 pays et se concentre sur les prêts avec une attention particulière aux personnes ayant peu ou pas d'antécédents de crédit.

Ceci couvre en grosso modo ce qu'il faut savoir sur le travail éffectué. Dans le deuxième chapitre nous nous intéressons à la définition et à la compréhension des concepts phares de la science des données qui sont mentionnés ou utilisés dans le reste du document. Ce chapitre vise à familiariser le lecteur avec ces notions avant de rentrer dans la partie technique du document, vous pouvez le voir comme une sorte de glossaire mais un peu plus détaillé.

Chapitre 2

La science des données

La science des données ou *data science* en anglais est un domaine des sciences à l'intersection des mathématiques et de l'informatique qui utilise des méthodes statistiques et des outils informatiques pour extraire et interprêter des informations et des patterns à partir des données. L'objectif est de répondre à des questions telles que : Que s'est il passé ? Pourquoi cela s'est produit ? Que va t-il se passer maintenant et ce qui peut être fait avec les résultats. Les scientifiques des données, plus communément appélés ***"data scientists"*** utilisent leurs découvertes pour aider les entreprises et autres organisations à prendre de meilleures décisions et à comprendre les évènements et les tendances.

Pour cela les data scientists sont souvent demandés de réaliser un mix de connaissances provenant de différents domaines des sciences comme les mathématiques, l'algorithmique et l'apprentissage automatique mais aussi d'avoir des connaissance du domaine où ils se spécialisent (**domain knowledge**) que ce soit la finance, l'éducation, la mode, le sport, tout domaine qui produit des données en soit.

La prediction des défauts de paiement est l'une des plus ancienne application de la science des données mais aussi l'une des plus complexe car c'est une tâche où les variables se complexifient d'années en années et où les tendances et les modèles deviennent assez vite obsolètes. Il est déjà commun pour les banques d'engager à temps plein des data scientists pour cette tâche entre autres. Il n'est pas non plus rares de voir des postes de data scientist dans la plupart des moyennes et grandes entreprises.

Dans ce chapitre, nous nous cherchons à construire une compréhension générale de la science des données, en définissant les différents concepts auxquels je risque de toucher dans la partie technique du document afin qu'à partir du chapitre 3 je puisse me concentrer uniquement sur l'explication des algorithmes et techniques implémentés et ainsi éviter de remplir les bas de page de notes.

2.1 Les données : Oxygène du nouveau monde

Depuis l'avènement du World Wide Web permettant aux individus du monde entier d'établir des rélations et de partager des connaissances, le phénomène du **big data** [1] a transformé notre monde. Chaque clic, chaque interaction, chaque transaction génère des données qui, une fois collectées et analysées, offrent une mine d'informations précieuses. Le domaine de la science des données a émergé pour exploiter ce potentiel.

La science des données et l'intelligence artificiel [2] ont permit de créer certains des outils le plus impactant du 21 ème siècle grâce à ces données.

Prenons par exemple une autre application classique de la science des données : les systèmes de recommendations. Les système de recommendations sont des modèles d'apprentissage automatique capable d'analyser et de suivre les préférences et intérêts des gens et de leur proposer des choses similaire (qu'ils sont donc susceptibles d'apprécier). Ces systèmes sont utilisés pour faire de la publicité ciblée ou par les réseaux sociaux pour proposer du contenu aux internautes. Ces systèmes ont atteint un tel niveau que beaucoup de gens sont convaincus d'être espionné. Si vous voulez connaître quelqu'un ou du moins savoir ce qui l'interêsse, prenez sont téléphone et ouvrez Youtube, facebook ou Tiktok pour les plus jeunes.

Parfois c'est éffrayant de réaliser tout ce que les grandes compagnies de la tech comme Google ou Facebook savent sur nous mais pourtant nous avons accepter les conditions d'utilisations et les règles de confidentialité. Vous connaissez surément l'expression : *"Internet n'oublie jamais"*, on ne peut plus être invisible aujourd'hui tant que l'on génère des données, chaque clic, chaque recherche, chaque contenu fournit des informations pouvant être utilisé pour savoir qui vous êtes, où vous êtes, ce que vous aimez ou pas et même ce que vous faites en ce moment même. C'est ça le nouveau monde, c'est dans ça que l'on vit et c'est ça le pouvoir et le science des données.

Après cette introduction, la prochaine section cherche à définir les différentes notions liées à cette science pour pouvoir transitionner de manière fluide vers la partie technique du mémoire.

1. Le terme "big data" fait référence à des ensembles de données si volumineux et complexes qu'ils dépassent les capacités des outils traditionnels de gestion et d'analyse. Ce phénomène a pris de l'ampleur avec l'explosion d'Internet et des réseaux sociaux. Chaque jour, des quintillions d'octets de données sont générés à partir de diverses sources : réseaux sociaux, sites web, capteurs, transactions financières, et plus encore.

2. L'intelligence artificiel est la science qui visent à créer des machines capables d'éfffectuer des tâches qui requièrent normalement de l'intelligence pour être accomplies ou capables d'agir d'une manière qui serait considérée intelligente si un humain agissait de la sorte

2.2 Concepts

Comme illustré dans "Figure 2.1", le flux de travail [3] typique d'un projet de science des données se compose de quatres grandes étapes :

1. Organisation : La première étape consiste à organiser les données de sorte à formuler un problème précis qui implique souvent de prédire la valeur d'une variable ou de classifier les exemples en divers catégories. Durant cette étape nous collectons et organisons les tables qui nous intéresse à partir de diverses sources de données pour former un ensemble de données (**dataset**)
2. Interprêtation : Durant cette étape on analyse l'ensemble de données avec des programmes qui implémentent des méthodes statistiques d'agregation et de transformation afin de créer un ensemble de prédicateurs qui peuvent être utilisés pour distinguer ou predire la variable qui nous intéresse.
3. Représentation : Durant cette étape nous cherchons à représenter les différentes variables sous une forme qui peut être comprise et manipulée par un ordinateur. Cela implique de représenter numériquement toutes les variables et de créer des tableaux, vecteurs ou matrices sur lesquels appliquer différents algorithmes et opérations mathématiques
4. Apprentissage : Durant cette étape nous créons des modèles d'apprentissage automatique qui apprenent à interprêter les rélations entre les différentes variables afin de créer une approximation de la fonction inconnue qui a généré les données d'entraînement et enfin utiliser cette approximation pour prédire la variable dépendante.

Dans les grands projets, ces différentes étapes sont typiquement exécutées par différentes personnes tous specialisé dans une des étapes. Un **Data Analyst** pour la première étape, un **Data Engineer** pour la séconde étape et un **Machine Learning Engineer** pour les deux dernières étapes. Un data scientist est une personne qui est capable d'exécuter le flux de travail de bout à bout, c'est-à-dire d'accomplir chacune de ces quatres étapes.

Dans mes recherches, j'ai exécuter ce flux de travail de la deuxième à la dernière étape, les données ayant déjà été collecter et organiser par Home Credit Group. Cela nécessite une connaissance des différents concepts impliqués dans chaque partie du flux de travail. La liste de définitions suivante couvre la plupart des notions fondamentales qui seront évoquées dans les prochains chapitres :

3. Un flux de travail est une série d'étapes impliquant des personnes et des systèmes qui sont effectuées afin d'atteindre un objectif au fil du temps. Autrement dit, c'est une succession de tâches qu'on effectue systematiquement pour faire un travail

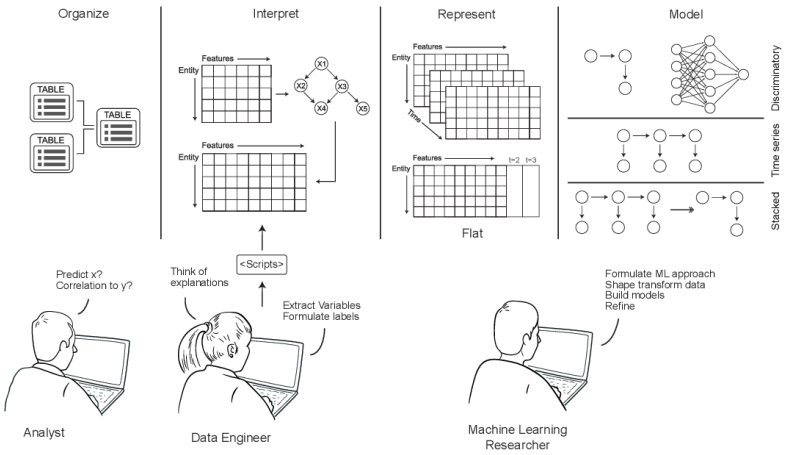

FIGURE 2.1 – Le flux de travail d'un projet de science des données et les rôles impliqués

— Une entité est une individualité possédant une existence et des caractéristiques propre. Dans ce document, une entité refère tout simplement à une table parmi celle de l'ensemble de données.
— Un prédicateur ou variable indépendante refère à une variable, une colonne dans une table pouvant être utiliser pour prédire une variable dépendante ou distinguer ces instances. Vous pourrez aussi entendre le mot anglais : **feature** pour faire référence à ce genre de variable. Par exemple la superficie d'une maison peut être utilisé pour prédire sa valeur.
— Une variable dépendante (**target**) est une variable/colonne dont on peut connaître la valeur en connaissant les valeurs d'une ou plusieurs variables indépendantes qui lui sont liées.
— Une instance est un cas particulier d'une entité où les différentes variables ou colonnes ont une valeur. On peut aussi parler d'exemple (eg : Un client, avec un nom, un genre, une addresse etc...).
— Une variable catégorique est un type de variable à valeur discrète et quantifiée (eg : La nationalité ou le sexe d'un client). À l'inverse une variable peut avoir des valeurs continues comme l'âge ou le salaire d'un client par exemple.
— La création ou l'ingenieurie de prédicateur est le processus qui permet de créer des variables indépendantes, généralement par aggrégation ou transformation de plusieurs variables de base.
— L'encodage est le processus par lequel on représente numériquement des variables qui ne sont pas de nature numérique (eg : sexe, ville...)
— Un entraînement réfère au processus lors duquel un model d'apprentissage automatique essaie de prédire la bonne valeur d'une variable dépen-

dante pour tous les exemples d'un ensemble d'exemples d'entraînement dont on connait déjà la valeur de la variable dépendante, corrigeant ses erreurs grâce à un algorithme d'apprentissage.
- Un algorithme est une séquence d'instructions reproducible et de calculs mathématiques visant à résoudre un problème spécifique.
- Un pipeline : Ici le mot pipeline fait référence à l'implémentation d'un flux de travail, c'est un suite ordonnée d'étapes (pouvant inclure plusieurs algorithmes chacun) organisée de sorte à pouvoir exécuter entièrement ou partiellement un flux de travail.

Ce chapitre était un peu la condition pour que ce mémoire soit accessible à un plus grand nombre de personnes avec l'idée de diminuer au maximum les prérequis nécéssaire à la compréhension. Dans ce court chapitre j'ai surtout cherché à introduire quelques notions de base auxquels je ferai référence ou que je developperai dans les prochains chapitres afin de m'assurer que les lecteurs d'autres parcours ou d'autres domaines puisse suivre. Ceci conclut l'introduction, dans la partie méthodologie je rentre dans les détails des différents algorithmes et techniques de la science des données que j'ai utilisé lors de mes recherches et leur impléméntations.

Deuxième partie

Méthodologie

Chapitre 3

Traitement des données

Avec l'intelligence artificielle en général et l'apprentissage automatique en particulier, les données sont toujours le nerf de la guerre, même plus important que les codes et les algorithmes les données d'entraînement détermineront à 80% le comportement du système. C'est donc tout naturellement la partie où l'on consacre le plus de temps et d'énergie. Avoir des données en quantité ne suffit pas, il s'agit de trouver les meilleurs prédicateurs, la meilleure représentation, la meilleure normalisation [1] etc...

Dans ce chapitre, j'explique les différents algorithmes et techniques de traitement des données que j'ai appliqué à l'ensemble de données original formé par Home Credit Group. Notez que ce chapitre ne couvre que les techniques que j'ai utilisé pour ce projet spécifique, il ne s'agit pas d'une introduction au traitement des données donc il sera plus concis et détaillé.

3.1 Création de prédicateurs

La création ou l'ingénieurie de prédicateurs (définis dans section 2.2) est très généralement la toute première tâche d'un data scientist et l'une des plus longue. Dans ce domaine les données sont rarement utilisables sous leur forme brute pour entraîner des modèles d'apprentissage automatique car elles proviennent généralement de diverses bases de données et sont constituées de plusieurs tables (liées ou indépendantes) représentant différentes entités et leur relations.

Ces données peuvent vite devenir extrêmement larges et complexes, dans les cas les plus extrêmes une seule entité de la base de données pourrait avoir

1. La normalisation refère au processus de transformation visant à reduire une colonne ou une table entière à une échelle jugée normale, par exemple faire un sorte que les valeur d'une colonne ne varie que entre zéro et un afin de les interprêter comme des probabilités.

des dizaines d'enfants [2] qui peuvent aussi avoir des enfants (donc des petits enfants pour l'entité de base). Sachant que la variable que l'on cherche à prédire est souvent une caractéristique de l'entité de base (comme la probabilité qu'un client revienne chez nous par exemple), ce genre de schéma complexifie le choix des prédicateurs puisqu'un prédicateur important peut être une caractéristique d'un enfant où d'un petit enfant (comme le nombre de commandes passées par le client ou le type de produits qu'il a commandé).

Avant de pouvoir créer des modèles de prédictions il est nécessaire de comprendre les relations entre les différentes entités et les caractéristiques intrinsèques de l'entité elle même pour en extraire des variables (*features*) pouvant nous permettre de faire le lien entre le comportement des entitées et la variable qu'on souhaite prédire (*target*).

Le langage des données peut être trivial des fois, par exemple supposons une table contenant uniquement les heures de travail hebdomadaire de chaque employé d'une entreprise. Un employé qui travaille plus est plus susceptible d'avoir une augmentation ou une promotion par exemple, donc si l'on cherche à prédire quels employés auront des promotions à la fin de l'année il est fort probable que cette information puisse nous aider, nous avons donc déjà une variable indépendante pour prédire les promotions.

Maintenant supposons qu'au lieu de cette table assez simple nous possédons une table qui contient les dates et les heures d'arrivées et de départs de chaque employé durant l'année. C'est moins évident tout d'un coup, sur quoi devons nous entraîner le modèle et comment représenter des dates et des heures ? Même si on trouve une représentation optimale (numérique) de ces variables, toutes ces variables sont-elles vraiment nécessaires pour prédire qui aura une promotion à la fin de l'année ? Elles répètent presque toutes la même information sous différentes formes, on risquerait d'entraîner un modèle inutilement complexe et qui risque de se noyer dans le bruit et la redondance (ralentissant ou stoppant complètement l'apprentissage).

Pourtant les données sur les arrivées et les départs des employés dans les entreprises sont typiquement plus fréquentes sous le deuxième format que le premier. En réalité, le premier est forcément déduit du second. En effet, nous pouvons passer de cette deuxième table à la première juste en appliquant des opérations de transformations et d'agrégations pour par exemple passer du format "date et heure" à "heure" seulement ensuite soustraire les heures d'arrivées des heures de départs (pour avoir les heures de travail journalier), faire la moyenne de ces heures de travail journalier sur des périodes de sept jours (semaines) et

2. Un enfant ici réfère à une entité qui a été créée par des opérations reproducibles éffectuées par ou sur une autre entité. Par exemple, dans un système de vente, une entité client pourrait avoir comme enfant une entité commande où l'entité commande contient toutes les commandes éffectuée par les différentes instances de l'entité client

enfin faire la moyenne de cette moyenne sur le nombre total de semaines travaillées pour avoir le nombre moyen d'heures travaillées par semaines au cours de l'année.

Ceci est un exemple plutôt simple illustrant un peu le processus général derrière de tâche de création de prédicateurs permettant de passer des données brute aux features ou prédicateurs.

Ce travail est effectué par un ingénieur en données (Data Engineer) avant le processus de création des modèles d'apprentissage. C'est une tâche fastidieuse dont la complexité augmente souvent avec le nombre de tables (entités) et selon la profondeur des rélations (enfant, petits enfants, arrière petits enfants...) dans la base de données et qui requière souvent une certaine expérience pour dénicher les meilleures features. Mais il s'agit aussi d'un travail qui devient vite extrêmement pénible quand l'ensemble de données brute que l'on possède dépasse une certaine taille et profondeur. L'ensembe de données avec lequel j'ai travaillé est constitué de **17 entités** dont l'entité de base (contenant des informations sur la demande de prêt et la variable à prédire) et 16 enfants de cette table de base. Certaines entités étaient tellement volumineuses qu'ils ont été obligés de les diviser en plusieurs tables, totalisant près de **70 tables** et **2588 colonnes** le tout regroupant des infomations tels que les demandes de prêt précedentes du client (s'il y'en a), son registre fiscal et d'autre informations personnelles et banquières sur les 1.526.659 démandeurs, certaines tables pouvant aller jusqu'à plus de 10 millions de lignes.

Alors je suis bien content qu'il n'y ait pas eu de petits enfants mais ça fait quand même une quantité colossal d'informations à analyser et à comprendre. Analyser toutes ces données afin de créer et évaluer des prédicateurs pertinents de façon manuelle me prendrait un temps énorme (risquerait d'aller jusqu'à un mois) or cette partie spécifique du flux de travail n'est pas ce qui nous intéresse le plus avec ce thème. Il me fallait trouver une solution beaucoup plus rapide, un moyen d'automatiser cette partie du travail.

3.1.1 Création automatique de prédicateurs

L'ingénieurie de prédicateurs est la partie du flux de travail qui nécéssite le plus l'intervention humaine donc les data scientist ont logiquement cherché à rendre ce travail moins long, surtout que la majeure partie du boulot consiste à écrire et modifier des fonctions pour appliquer les même calculs mathématiques à différentes table avec différents schémas. Il s'agissait de trouver un niveau d'abstraction qui s'approche au maximum de la compréhension humaine de cette tâche et qui puisse permettre la création d'algorithmes capable de prendre en charge ce processus.

Néanmoins, la création automatique de prédicateurs, faisant référence à des

algorithmes capable de prendre en charge entièrement la création de prédicateur, est un concept plus ou moins récent dans le domaine de la science des données . Le concept fût d'abord formalisé par James Max Kanter et Kalyan Veeramachaneni dans leur papier de 2015 "Deep feature synthesis : Towards automating data science endeavors" où ils proposent un algorithme pour générer automatiquement des prédicateurs pour les ensembles de données relationnelles : c'est l'algorithme de Deep Feature Synthesis ou Synthèse approfondie des prédicateurs (KANTER et VEERAMACHANENI, 2015).

L'algorithme suit les relations entre les différentes tables et une table de base, puis applique séquentiellement des fonctions mathématiques d'aggrégation et de transformation le long de ces chemins, en fonction du type de données des colonnes (nombres, booléens, chaînes de caractères, horodatages etc...), pour enfin créer des prédicateurs. L'algorithme prend en entrée un ensemble interconnecté d'entités/tables.

Dans le papier, les auteurs ont décrit deux types de relations entre entités, les relations vers l'avant et les relations vers l'arrière (*forward and backward relationships*), et trois types de prédicateurs pour une entité en fonction des relations qu'elle entretient avec d'autres entités : les prédicateurs d'entité (**efeat**), les prédicateurs directes (**dfeat**) et les prédicateurs relationnelles (**rfeat**).

Dans une relation vers l'avant, une instance d'une entité E^l est associée à une seule instance d'une autre entité E^k. Et une relation vers l'arrière est une relation qui lie une instance i dans E^k à toutes les instances m = 1 . . . M dans E^l qui ont une relation vers l'avant avec k. L'exemple trivial est celui d'une relation "parent-enfants" où un enfant n'a basiquement qu'un parent alors que le parent peut avoir beaucoup d'enfants. D'ailleurs les scientifiques ont majoritairement adopté cette appellation.

1. Les prédicateurs d'entité (efeat) : ce sont les caractéristiques directes de l'entité (les variable dans la table de base de l'entité) comme le sexe, l'âge, le revenu, etc... En fonction de la variable à prédire, ce type feature pourrait même être utilisé directement comme prédicateurs. On peut aussi leur appliquer des transformations, par exemple : calculer l'âge à partir de la date de naissance.
2. Les prédicateurs directes (dfeat) : les prédicateurs directes sont appliquées sur les relations vers l'avant (Enfant -> Parent). Dans ceux-ci, des features de l'entité i $\in E^k$ (enfant) sont directement transférées sous forme de caractéristiques pour l'entité j $\in E^l$ (parent). Un exemple pourrait être le prix d'un produit qui devient un attribut dans la table des commandes.
3. Les prédicateurs relationnelles (rfeat) : les prédicateurs relationnelles sont appliquées aux relations vers l'arrière (Parent -> Enfants). Ils sont dérivés pour une instance i de l'entité E^k en appliquant une fonction mathéma-

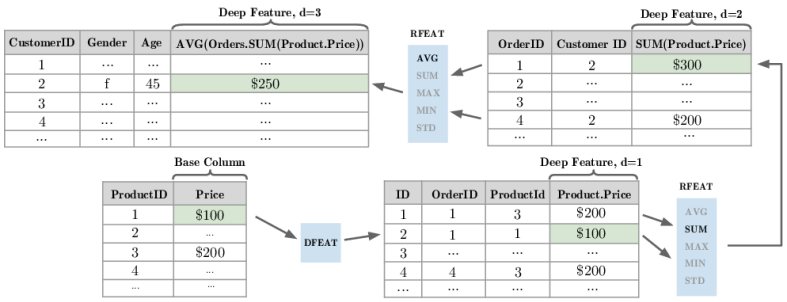

FIGURE 3.1 – Visualisation d'une itération de la Synthèse approfondie de prédicateurs (DFS)

tique à $x^l_{:,j|ek=i}$, qui est une collection de valeurs pour une caractéristique j dans l'entité E^l assemblée en extrayant toutes les valeurs de j dans l'entité E^l où l'identifiant de E^k est $ek = i$. Cette transformation est donnée par la formule $x^k_{i,j'} = rfeat(x^l_{:,j|ek=i})$. Quelques exemples de fonctions rfeat sont min (pour trouver le minimun d'un ensemble de valeur), max (pour trouver le maximumn d'un ensemble de valeur) et count (pour compter les apparitions d'une valeur en particulier).

Avec ce niveau d'abstraction, le but de l'algorithme de Deep Feature Synthesis est de trouver et de calculer les dfeats et les rfeats pour une entité cible E^k, les ajoutez à la table E^k et calculez les efeats sur l'ensemble. Dans le cas où une entité enfant de l'entité cible possède ses propres enfants, nous pouvons générer les prédicateurs de manière récursive en utilisant la même séquence décrite ci-dessus. La récursion peut se terminer lorsque l'on atteint une certaine profondeur (eg, arrière petits enfants) ou s'il n'y a plus d'enfants.

Ainsi Kanter et Veeramachaneni ont proposé et implémenté un algorithme qui possède un niveau d'abstraction suffisant pour imiter l'intuition humaine pour la tâche de création de prédicateurs. La figure 3.1 apporte une illustration de l'algorithme

3.1.2 Implémentation

J'ai donc utilisé l'agorithme de Synthèse approfondie de prédicateurs sur les données fournies par Home Credit Group. Pour cela j'ai utilisé le langage de programmation python et deux bibliothèques entre autres que j'aimerai souligner : **"featuretools"** qui offre une implémentation de DFS et **"Dask"** pour paralléliser les calculs (DASK DEVELOPMENT TEAM, 2016).

Au vue de la taille de l'ensemble de données fournit par Home Credit j'ai été obligé d'implémenter quelques autres stratégies, notamment pour partitionner

les données, car trop volumineuse pour pouvoir être procédé directement même avec 30 Go de mémoire vive. L'ensemble de données a donc été partitionnée de manière à ce que chaque partition contienne exactement 17 tables (1 entité == 1 table) avec toutes les informations nécéssaires pour faire la synthèse des prédicateurs pour un sous ensemble de maximum 12.000 exemples (demandeurs). Résultant en 128 partitions de 17 tables parfaitement indépendantes (les partitions sont indépendantes entre elles mais les tables dans la même partition sont liées).

L'utilisation de Dask pour paralléliser les opérations a permit d'appliquer DFS sur plusieurs partitions en même temps (4 pour être précis, correspondant au nombre de cœurs du processeur) et donc calculer plusieurs matrices de prédicateurs simultanément, diminuant le temps d'exécution du code de 44-50 heures à approximativement 24 heures. Le code en question est librement accessible sur Kaggle et github (Diarra, 2024b).

3.2 Sélection de prédicateurs

Après l'étape précédente de création de prédicateurs nous sommes passé d'un ensemble de près de 70 tables interconnectées contenant plusieurs type d'informations brutes à 128 tables indépendantes contenant 2526 prédicateurs potentiels et leur valeurs pour maximum 12.000 demandeurs (128 étant le nombre de partitions créées). Par contre, il est préférable de ne pas utiliser autant de prédicteurs pour entraîner des modèles, enfaîte c'est même une très mauvaise.

Comme je l'ai expliqué dans la précédente section, l'algorithme de synthèse approfondie de prédicateurs cherche à imiter l'intuition des data scientists en appliquant sequentiellement les mêmes fonctions mathématiques utilisées par ces derniers sur une relation ou une caracteristique brute lorsque les types des données sont compatibles avec ces opérations. Mais cette intuition n'est pas tout le temps correcte, des scientifiques expérimentés peuvent se tromper sur le pouvoir de prédiction d'un prédicateur potentiel alors un algorithme encore plus. Avec Deep Feature Synthesis nous misons sur la quantité et dans cette quantité il y'aura forcément quelques bon voir très bon prédicateurs, même souvent des prédicateurs auxquels un humain n'aurait pas du tout pu penser en raison de leur profondeur ou de leur nature mais une grande majorité des prédicateurs générés par l'algorithme seront redondants, peu informatifs ou même complètement absurdes.

Une étape importante qui suit systematiquement la création de prédicateurs est celle de **sélection de prédicateurs** (feature selection). Cette étape se caractérise par deux grandes approches inspirées de deux philosophies différentes, sélectionner les meilleurs ou éliminer les plus mauvais, les deux reviennent sou-

vent au même résultat. La première méthode ou méthode d'emballage (*wrapper method*) consiste à diviser les prédicateurs en sous groupes, entraîner un modèle avec chaque sous groupe et sélectionner le sous groupe surlequel a été entraîner le meilleur modèle, l'idée étant de donner une chance à chaque prédicateur potentiel et de laisser le modèle lui même déterminer et choisir les meilleurs. Cette méthode est généralement utilisée quand l'on possède un ensemble de prédicateurs que l'on consideère déjà comme bon (dont on connaît le pouvoir de prédiction) et qu'on veut réduire la taille et la complexité du modèle en sélectionnant les meilleurs sans trop affecter les perfomances du modèle. Elle possède par contre l'inconvénient du temps, si vous possédez beaucoup de prédicateur comme dans notre cas (qu'on ne peut pas tous appeler de bon prédicateurs en plus) cette méthode prendra le temps d'évaluer chaque sous groupe après entraînement ce qui va prendre beaucoup de temps, de plus si les sous groupes sont crées aléatoirement il est très probable qu'un sous groupe mélange des bons et mauvais prédicateurs et soit quand même sélectionné comme le meilleur.

La deuxième méthode ou méthode par filtrage (*filter method*) a un principe de fonctionnement différent, éliminer les plus mauvais prédicateurs en fonction d'un ou plusieurs critères statistiques et arbitraires. Elle est beaucoup plus adaptée à notre situation car nécéssitant moins de temps et nous permettra de se débarasser des erreurs générées par l'algorithme. C'est donc celle que j'ai utilisé. Il est techniquement possible de combiner les deux méthodes, c'est-à-dire de filtrer les plus mauvais et ensuite de sélectionner les meilleurs des meilleurs, mais il existe aussi une méthode par filtrage qui peut être utiliser pour selectionner les meilleurs prédicateurs en se basant sur le gain d'information que j'ai utilisé et donc je n'ai pas juger nécéssaire d'utiliser d'autres méthodes d'emballage ensuite. Dans la prochaine sous-section je décris les techniques de filtrage que j'ai implémenté.

3.2.1 Méthodes de filtrage

J'ai implémenté quatre techniques de filtrage avec l'objectif de diminuer le nombre de prédicateurs de 2526 aux 100 plus informatifs (le nombre 100 a été choisi arbirairement et ne répond donc à aucune convention ou calcul préalable).

1. **Supprimer les colonnes/prédicateurs avec beaucoup de valeurs manquantes** : Dans une entité, un attribut peut ne pas être strictement nécéssaire pour définir l'entité, c'est-à-dire qu'une instance de l'entité peut ne pas avoir de valeur pour cet attribut. Zéro étant considéré comme une valeur, il existe communément dans les bases de données une constante appélée **null ou none** représentant les cas où aucune valeur n'est applicable pour cette variable (signifiant que l'instance en question ne pos-

sède pas cet attribut). Par exemple, une personne pourrait avoir comme profession "none" (signifiant qu'elle est au chômage). Dans une tâche de prédiction maintenant si il y'a des prédicateurs pour lesquels la majorités des exemples ne possèdent pas de valeur, cela signifie que ce prédicateur n'est pas assez important pour déterminer la variable à prédire donc on peut s'en débarasser.

2. **Supprimer les colonnes/prédicateurs avec une faible variance** : De manière analogue au cas précédent, un prédicateur qui ne varie pas d'un exemple à un autre ne peut nous aider à distinguer les exemples. Si presque tous les élèves dans une classe ont comme nom de famille **Traoré**, il ne me sert à rien de savoir que Traoré a triché à l'examen si je veux savoir qui a triché, ce qui est sûr c'est Traoré. La variance est une mésure statistique de la différence entre les valeurs dans une distribution. Plus précisément, elle mesure la distance entre chaque nombre de l'ensemble et la moyenne, et donc de tous les autres nombres de l'ensemble. Éliminer ces prédicateurs est un moyen de se débarasser des prédicateurs peu ou pas du tout infomatifs. La variance est le carré de l'écart type et est définie par la formule :

$$V = \frac{\Sigma_{i=1}^{N}(x_i - \mu)^2}{N}$$

où N est le nombre total d'exemples et μ la moyenne de la distribution.

3. **Supprimer les colonnes/prédicateurs avec une trop forte corrélation avec une autre colonne** : Il peut également arriver qu'un prédicateur soit informatif mais répétitif car un autre fournit presque excatement la même information. Par exemple il serait inutile d'avoir un prédicateur dont la valeur est égale au revenu mensuel d'un client si il y'a déjà un predicateur qui offre des informations sur les revenu hebdomadaire du client, il est sûr qu'un revenu hebdomadaire plus ou moins grand donnera un revenu mensuel proportionnel. On dit que le revenu hebdomadaire a une corrélation absolue parfaite (égale à 1) avec le revenu mensuel car augmenter ou diminuer le premier va augmenter ou diminuer le second dans 100% des cas. La corrélation est une mesure statistique qui exprime à quel point deux variables sont linéairement liées. Le coefficient de corrélation entre deux variables x et y est donnée par la formule :

$$Corr = \frac{\Sigma_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{N}(x_i - \bar{x})^2 \cdot \Sigma_{i=1}^{N}(y_i - \bar{y})^2}}$$

où N est le nombre total d'exemples, $\bar{x}$ la moyenne de la variable x et $\bar{y}$ la moyenne de la variable y.

4. **Sélectionner les 100 plus informatifs prédicateurs** : Dans une tâche de prédiction tous les prédicateurs n'ont pas le même impact sur la variable à prédire. La corrélation linéaire pourrait être utilisée pour savoir quels prédicateurs a la plus grande importance, en calculant le coefficient de la relation linéaire entre chaque prédicateur et le target, celui qui a la plus grande corrélation serait le prédicateurs le plus informatif. Mais en pratique toutes les rélations entre les prédicateurs et la variable dépendante ne sont pas aussi simple que la corrélation, dans les ensemble de données plus élaborés les variables peuvent partager des rélations plus complexes qui ne sont même pas d'ordre linéaire. Un critère plus général serait le **gain d'information**. Le gain d'informations d'un prédicateur est une mesure de la capacité de ce prédicateur à distinguer les exemples, plus exactement à quel point connaître la valeur de ce prédicateur nous permet de connaître la valeur de la variable dépendante et il est calculé grâce à l'entropie. L'entropie est la grandeur fondamentale de la théorie de l'information, c'est une mesure de l'incertitude d'une variable aléatoire ; plus il y a d'informations, moins il y a d'entropie. Le gain d'informations d'un attribut est calculé comme la différence entre l'entropie de l'ensemble des données avant que l'on considère cet attribut et l'entropie de l'ensemble de données une fois la valeur de l'attribut connue.

$$IG(D, A) = H(D) - H(D|A)$$

où $H(D)$ est l'entropie de l'ensemble de données avant et $H(D|A)$ est l'entropie pondérée après. En général, l'entropie d'une variable aléatoire V avec des valeurs v_i ayant une probabilité $P(v_i)$ est définie comme :

$$Entropy\ :\ H(V) = \Sigma_i P(v_i) \cdot log_2(\frac{1}{P(v_i)}) = -\Sigma_i P(v_i) \cdot log_2(P(v_i))$$

Et l'entropie pondérée est donnée par la formule :

$$H(D|A) = \Sigma_i w_i \cdot H(D_i)$$

où w_i est le nombre d'exemples dans le sous-ensemble D_i par rapport à l'ensemble de données entier (w_i = nombre d'exemples où $A = v_i$ / nombre total d'exemples dans D) et $H(D_i)$ est l'entropie du sous-ensemble D_i.

3.2.2 Implémentation

J'ai écrit une fonction pour chacune des techniques expliquée ci dessus, toujours dans le même langage de programmation, un seuil a arbitrairement été choisi pour les trois premières techniques. En raison de taille de l'ensemble de

données encore une fois, j'ai utilisé une stratégie simple, pour ne pas puiser mes ressources et augmenter le temps d'exécution du code, j'ai combiner les matrices de prédicateurs de 32 des 128 partitions soit à peu près 25% des exemples et je n'ai appliquer les différentes fonctions que sur ce sous-ensemble. J'ai ensuite généraliser les résultats à l'ensemble de données entier en reportant les mêmes prédicateurs sélectionnés. Reduisant ainsi le nombre de prédicateurs de 2526 à 100, ce qui permettra d'accélérer les entraînements et de simplifier l'interprêtation des modèles (Diarra, 2024d).

Quelques uns des prédicateurs sélectionnés sont : le niveau d'étude du demandeur, ses types de revenues, les langues qu'elle parle, les rôle qu'elle a occupés dans sa vie au niveau professionnel, son sexe, le nombre de demandes de prêts passés, le nombre de jours de retard de délai (s'il y'a eu retard), le mois, la saison et le jour de la semaine de la decision finale (si oui ou non le prêt lui avait été accordé), sa residences, ses charges financières etc... Ce nouvel ensemble de données a aussi été publié (Diarra, 2024c).

3.3 Prétraitement des données

C'est la dernière étape du traitement des données, à cette étape du pipeline nous cherchons une représentation finale pour les données un format directement utilisable lors de l'entraînement, des tests et du déploiement. Traditionnellemnt c'est l'étape qui précède directement l'entraînement des modèles. Généralement, elle consiste en trois étapes :

1. **Gérer les valeurs manquantes** : Lors de la selection des prédicateurs on a éliminé les colonnes contenant plus de 70% de valeurs manquantes. Des colonnes pourrait être en dessous de ce seuil mais contenir un nombre assez important de valeurs manquante. Pour ne pas avoir à supprimer les examples à qui appartiennent les valeurs manquantes, il est nécéssaire de trouver une réprésentation pour ces valeurs manquantes. Ma priorité était de garder tous les exemples donc les valeurs manquantes on été remplacées par deux valeurs volontairement hors de la distribution (**-9999** pour les valeurs colonnes de type numérique et le token "<N/A>" pour les prédicateurs non numériques.
2. **Encodage** : Comme défini dans la section 2.2, l'encodage est une étape où l'on cherche une représentation numérique pour toutes les variables qui ne le sont pas par nature. Au vue de la grande dimensionnalité des données que l'on possède, j'ai opté pour la forme la plus simple d'encodage possible en attribuant un identifiant numérique unique (dans sa colonne) à chaque variable de type catégorique, jour de la semaine, mois, saison ou chaîne de caractère. Mais les valeurs sont déterminées par l'ordre d'occurence

des différentes valeurs dans le sous-ensemble de référence. Par exemple, si la première saison à apparaître dans le sous ensemble de référence est l'automne alors l'automne sera représenter comme 0 et ainsi de suite.

3. **Normalisation** : La normalisation standard consiste à transformer les colonnes de manière à ce que leur valeurs décrivent une distribution de moyenne 0 et d'écart type 1. Pour cela on peut calculer la moyenne et la variance de chaque colonne et on remplace chaque valeur grâce à la formule

$$x = \frac{(x - \bar{x})}{\sqrt{Variance}} = \frac{(x - \bar{x})}{\sigma}$$

Enfin il est important de noter que les données sont aléatoirement divisées en deux groupes distincts avant l'entraînement, un ensemble d'entraînement et un ensemble de test. L'ensemble de test, prévu pour évaluer le modèle après entraînement correspond à 5% des 1.526.659 demandes.

Chapitre 4

L'apprentissage automatique

Dans le domaine de l'intelligence artificielle, on dit qu'un agent apprend quand ses performances s'améliorent après quelques observations sur le monde. Lorsque l'agent est une machine, on appelle les observations « **data** » ou données en français et le processus « apprentissage automatique » ou apprentissage machine.

Dans l'apprentissage automatique, un ordinateur observe certaines données, construit un modèle basé sur ces données et utilise le modèle à la fois comme hypothèse sur le monde et comme logiciel capable de résoudre des problèmes (Russell et Norvig, 2020). Le machine learning désigne donc une famille de techniques et d'algorithmes d'apprentissage basés sur l'induction [1] utilisés pour que les machines puisse extraire des « connaissances » à partir de données.

Ce chapitre n'est pas prévu d'être une introduction à l'apprentissage automatique, je n'explique ici que les algorithmes derrière les cinq modèles que j'ai développé pour cette tâche, à savoir la régression logistique, la forêt aléatoire, le boost de gradient, les réseaux neuronaux artificiels et les machines à vecteur de support. Cependant il n'assume pas non plus de connaissance préalable car les différents algorithmes sont expliqués en profondeur.

4.1 Introduction à l'apprentissage supervisé

Il y'a plusieurs types d'apprentissage dans l'apprentissage automatique, chacune avec ses propres familles d'algorithmes, mais le processus d'apprendre à partir d'exemples dont on connaît déjà les valeurs de la variable dépendante est

1. L'induction est l'opposé de la déduction. La déduction consiste à tirer des conclusions spécifiques à partir de prémisses « générales » attestées, les conclusions tirées de la déduction sont garanties comme vraies tandis que l'induction vise à développer des conclusions générales à partir d'exemples spécifiques et peut donc conduire à des conclusions incorrectes.

ce que l'on appelle : **Apprentissage supervisé**.

Dans l'apprentissage supervisé, l'agent observe les paires entrée-sortie et apprend une fonction qui mappe l'entrée à la sortie. Par exemple, les entrées pourraient être des images de caméra, chacune accompagnée d'une étiquette (label) indiquant « bus » ou « piéton », etc. Le modèle doit apprendre une fonction qui, lorsqu'on lui donne une nouvelle image, prédit l'étiquette appropriée. En d'autres termes, étant donné des exemples de paires d'entrées-sorties (x, y) générés par une fonction inconnue f, où $y = f(x)$, la tâche de l'apprentissage supervisé est de trouver une fonction h qui se rapproche si étroitement de la vraie fonction f que nous pourrions utiliser h pour prédire les comportements futurs de f.

Les données utilisées pour trouver la fonction h sont appelées **"ensemble d'entraînement"**. Nous disons que le modèle d'apprentissage automatique généralise bien s'il prédit avec précision les étiquettes d'un autre ensemble de données inédites appelé **ensemble de test**.

Il existe essentiellement deux types de problèmes dans l'apprentissage supervisé selon le type d'étiquette : lorsque le résultat fait partie d'un ensemble fini de valeurs telles que ensoleillé/nuageux/pluvieux (prédire la météo) ou vrai/faux, le problème d'apprentissage est appelé **classification**. Lorsqu'il s'agit d'un nombre continu (comme la température de demain, mesurée soit sous forme d'entier, soit sous forme de nombre réel), le problème d'apprentissage porte le nom (certes étrange) de **régression**. Notre problème ici est de prédire quels clients vont faire défaut sur un prêt, c'est donc une **classification binaire**.

Il existe également deux types de modèles d'apprentissage automatique : les modèles paramétriques et les modèles non paramétriques. Les modèles paramétriques (préférés pour les très grands ensembles de données) résument toutes les connaissances de l'ensemble de formation dans un ensemble relativement petit de paramètres ajustables (souvent appelés **poids**). Les exemples incluent les modèles de régression linéaire et les réseaux de neurones. Les modèles non paramétriques (plus simple pour les ensembles de données plus petits) utilisent l'ensemble d'entraînement lui-même pour créer leur hypothèse, ils conservent souvent les données d'entraînement ou au moins une partie d'elles en mémoire et les utilisent à des fins de prédiction. Les exemples incluent l'algorithme des K plus proches voisins (**K-Nearest Neighbors**) et les arbres de décision. Il existe également des modèles qui combinent les deux philosophies, on pourrait les appeler : *"Modèles Hybrides"*. Dans les prochaines sections on cherche à avoir une compréhension des différents algorithmes d'apprentissage utilisés dans mes recherche et du fonctionnement des modèles qu'ils créent.

4.2 Les arbres de décisions

Un arbre de décision est une représentation d'un processus de prise de décision séquentiel qui prend un vecteur de paires **attribut-valeur** et renvoie une sortie unique, une « décision ». Un arbre de décision est un modèle non paramétrique qui prend sa décision en effectuant une séquence de tests sur les valeurs des attributs d'un exemple donné, en commençant par la racine et en suivant la branche appropriée jusqu'à atteindre une feuille (fin d'un chemin). Chaque nœud interne de l'arborescence correspond à un test de la valeur de l'un des attributs d'entrée, les branches du nœud sont étiquetées avec les valeurs possibles de l'attribut, et les feuilles précisent quelle valeur doit être renvoyée par la fonction à la fin d'un chemin donné.

Les valeurs d'entrée peuvent être discrètes ou continues, même si les arbres de décision sont généralement utilisés avec des attributs/prédicateurs catégoriques, nous pouvons tirer parti du seuillage ou de tout autre type d'encodage pour utiliser des attributs à valeur continue comme entrée. Par exemple, âge entre 0 et 40 => catégorie-1, âge > 40 => catégorie-2). Cependant, les résultats ou les décisions doivent provenir d'un ensemble fixe de valeurs, appélées des **classes**. Lorsque les classes sont simplement vrai/faux ou rembousera/remboursera-pas (le prêt), nous appelons cela : classification booléenne ou binaire.

Un arbre de décision peut techniquement renvoyer des valeurs continue, on les appelle **arbres de régression**, ils fonctionnent de la même manière que les arbres de décision, dans le sens où ils suivent un chemin en fonction des valeurs des attributs mais au lieu de continuer ce processus jusqu'à ce qu'ils atteignent une « feuille », ils renvoient une prédiction qui est la moyenne des étiquettes dans le dernier nœud. Par exemple, si les autres points de données qui ont les mêmes valeurs d'attribut que celui pour lequel nous voulons prédire une étiquette ont les étiquettes suivantes : 7, 7, 10, 9. L'arbre de régression renverra alors $\frac{7+7+10+9}{4} = 8.25$. Dans certains cas il n'y a tout simplement plus d'attributs pour différencier les exemples, dans ce genre de cas un arbre de décision peut agir comme un arbre de régression ou juste retourner l'étiquette la plus fréquente dans le dernier nœud, avec l'exemple précédent un arbre de décision retournerait 7.

Un arbre de décision est donc équivalent à l'expression logique suivante :

$$Sortie = (chemin_1 \vee chemin_2 \vee chemin_3 ...)$$

où chaque $chemin_i$ est une conjonction de la forme $(A_m = v_x \wedge A_n = v_y \wedge ...)$ de tests de valeur d'attribut correspondant à un chemin de la racine à une feuille. Ainsi, toute fonction en logique propositionnelle peut être exprimée sous forme d'arbre de décision.

Ok, un arbre de décision prend une décision en suivant un chemin décidé par des tests sur les valeurs des attributs d'un exemple, mais comment construire ces

ID	SEX	PNE	PRE	AGE	DIA	LAB
0	2	1	2	73	1	2
1	1	2	2	39	2	0
2	2	1	2	46	2	1
3	1	1	2	74	2	2
4	2	1	2	1	2	1
5	1	2	2	46	2	0
6	1	2	2	24	2	0
7	2	2	2	54	1	2
8	1	1	1	37	2	1
9	1	2	1	42	2	0

TABLE 4.1 – Données synthétiques (Covid Classification)

chemins ? En fait, le problème n'est pas de savoir comment construire un arbre de décision mais comment « l'apprendre » à partir des données. En effet, le problème est que si nous voulons simplement un arbre de décision, il n'existe pas qu'une seule façon de le construire et il n'y a pas qu'un seul arbre qui corresponde à un ensemble de données donné. Par exemple, si nous devions programmer un arbre de décision, nous pourrions utiliser n'importe quel attribut de notre ensemble de données comme racine et diviser les données selon n'importe quel aspect ou critère qui nous intéresse. Cela pourrait donner lieu à des centaines d'arbres possibles pour un ensemble de données, mais tous ne sont pas de « bons arbres de décision ».

4.2.1 Construire un arbre de décision par intuition

Pour l'illustration, considérons l'ensemble de données présenté dans le tableau 4.1. Cet ensemble est constitué de cinq attributs ou prédicateurs, le but étant de prédire le risque d'un patient atteint du covid-19 avec trois classes : SOFT_COVID (le patient devrait aller bien et a besoin de moins d'attention), STRONG_COVID (le patient est en danger et a besoin de plus d'attention) et DEAD (le patient doit être considéré comme une urgence et pourrait mourir). Ce petit ensemble de données n'est là que pour illustrer le fonctionnement des arbres de decisions et expliquer les concepts, je l'ai choisi pour cet exemple parce que toutes les variables sont catégoriques ce qui va simplifier la représentation de l'arbre, il a été générer par un réseau de neuronnes artificiel que j'ai entraîner sur d'autres données du monde réel (DIARRA, 2023). Pour simplifier on considerera juste dix exemples d'entraînements et cinq attributs, cela permettra de pouvoir faire les calculs à la main.

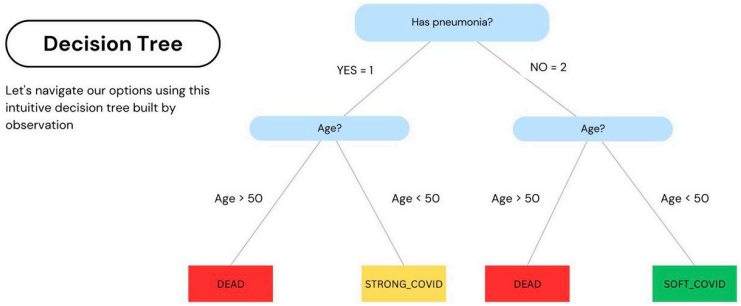

FIGURE 4.1 – Un arbre de décision construit en choisissant l'attribut qui divise l'ensemble de données en deux sous-ensemble de taille égale comme racine

Signification des prédicateurs :

- SEX : 1=femme, 2=homme
- PNEUMONIE (PNE) : si le patient présente déjà une inflammation pulmonaire ou non
- ENCEINTE (PRE) : si la patiente est enceinte ou non
- ÂGE : l'âge du patient
- DIABÈTE (DIA) : si le patient est diabétique ou non
- PRÉDICTION (LAB) : 0 (SOFT_COVID), 1 (STRONG_COVID), 2 (DEAD).

Un arbre de décision pour ce problème de classification est la représentation de n'importe quelle fonction logique capable de distinguer ces exemples. N'importe quel attribut de ce tableau peut être la racine d'un arbre (une solution), nous pouvons donc facilement construire un arbre qui classifie correctement ces exemples. Un arbre de décision possible pour ce problème de classification de covid est montré dans la figure 4.1.

Avec l'arbre montré dans la figure 4.1, on s'est arrêté parce qu'il ne restait plus aucun exemple d'entraînement qui ne soit pas classifiable avec cet arbre mais en pratique on peut toujours ajouter des nœuds. Simplement construire un arbre de décision n'est donc pas compliqué. Le problème est que nous avons construit cet arbre dans le but de classifier juste ces dix exemples d'entraînements or notre but est d'obtenir un modèle qui est appris à généraliser des connaissances tirées du jeu de données pour pouvoir prédire les étiquettes d'autres exemples, c'est cela la tâche de l'apprentissage supervisé, il s'agit de se rapprocher au maximum de la fonction inconnue qui a générée les données (dans ce cas précis un réseau de neuronnes artificiel) pas de mémoriser l'ensemble d'entraînement.

4.2.2 Apprendre un arbre de décision

L'essence de l'apprentissage dans les arbres de décision réside dans la création d'un modèle capable de prédire avec précision les étiquettes de nouvelles données invisibles tout en restant simple et relativement petit (RUSSELL et NORVIG, 2020). Apprendre un arbre de décision consiste à trouver l'attribut le plus important pour diviser les données à chaque étape, de la racine au dernier nœud. Par « attribut le plus important », nous entendons celui qui fait le plus de différence dans la classification d'un exemple. En termes simples, l'attribut le plus important d'un ensemble de données est celui qui divise l'ensemble de données en groupes aussi homogènes que possible de part ses valeurs. Traditionnellement on utilise le concept de gain d'information expliqué dans la section 3.1.1 pour déterminer l'attribut le important à chaque nœud, même si toute autre fonction pourrait être utiliser. L'attribut le plus important est donc une autre appelation de l'attribut le plus informatifs.

4.2.3 Algorithme

Une fois que l'on a trouvé l'attribut le plus informatif, on divise l'ensemble de données en sous-ensembles en fonction des valeurs de cet attribut. C'est une mannière de diviser recursivement le problème de base en plus petits sous problèmes pour lesquels on peut appliquer l'algorithme suivant :

1. Si les exemples restants sont tous dans la même catégorie (de la même classe), alors nous avons terminé : nous pouvons renvoyer cette classe comme décision.
2. Sinon Si les exemples restants sont hétérogènes (de classes différentes), choisir le meilleur attribut pour créer un nouveau nœud.
3. S'il ne reste plus d'exemples, cela signifie qu'aucun exemple n'a été observé pour cette combinaison de valeurs d'attribut, et nous renvoyons la valeur de sortie la plus courante de l'ensemble d'exemples utilisés lors de la construction du parent du nœud (nous pouvons également implémenter n'importe quel autre logique dans ce cas).
4. S'il ne reste plus d'attributs, mais des exemples de classes différentes, cela signifie que ces exemples ont exactement la même description, mais des classifications différentes. Cela peut se produire en raison d'une erreur ; parce que l'environnement est non déterministe ; ou parce que nous n'avons pas ou ne pouvons pas observer un attribut qui distinguerait les exemples. Le mieux que nous puissions faire est de renvoyer la valeur de sortie la plus courante des exemples restants.

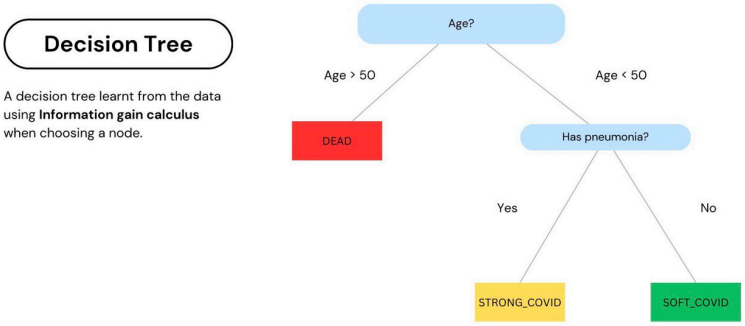

FIGURE 4.2 – Un arbre de décision construit grâce au gain d'information

En appliquant l'algorithme ci dessus aux données de la table 4.1 on obtient l'arbre illustré dans la figure 4.2. La principale différence entre cet arbre et l'arbre de la figure 4.1 est que celui là est construit en fonction du gain d'information et est donc capable de changer pour s'adapter à d'autres ensemble de données ou simplement un plus grand ensemble. On peut donc dire que cet algorithme peut « apprendre » des données. Pour la tâche de prédiction des défauts de paiement je n'ai pas directement implémenté d'arbres de décisions mais d'autres algorithmes basés sur les arbres de décisions, donc une compréhension des arbres de décisions (offerte dans cette section) me permettra d'être plus concis dans l'explication des algorithmes de la Forêt aléatoire et du boost de gradient.

4.3 La forêt aléatoire

La forêt aléatoire est un algorithme de classification basé sur les arbres de décisions. Un aspect importants des arbres de décisions et des modèles non paramétriques en général est que ce sont les modèles les plus sensibles au sur-ajustement [2] ou **overfitting**. Comme son nom laisse entendre, l'idée derrière l'algorithme de la forêt aléatoire de est de combiner ou de sommer les prédictions de plusieurs arbres de décisions entraînés sur différents sous groupes (créés aléatoirement) de l'ensemble d'entraînement (BREIMAN, 2001). C'est la philosophie des **techniques d'ensemble** de *Bagging* qui cherche à combiner ou confronter les prédictions de plusieurs apprenants faibles ou **weak learners** (traditionnellement des arbres de décisions) en espérant que leur divergence corrige leur bias

2. Le sur-ajustement est le nom du phénomène qui se produit quand un modèle d'apprentissage n'arrive pas à généraliser mais excelle à prédire les données d'entraînement. C'est un signe que le modèle est trop dépendant de la structure et des exemples de l'ensemble d'entraînement, on dit donc qu'il est sur-ajuster sur les données d'entraîment

respectifs. Cela resulte en des modèles moins sensible au sur-ajustement et donc une meilleur capacité de généralisation puisque la classification ne dépend plus seulement d'un arbre potentiellement sur-ajusté mais de toute une forêt avec des individus de différents schéma, perception et bias.

4.3.1 Algorithme

Bien que les implémentations peuvent différer, l'algorithme général se présente comme suit :

1. **Créer des sous-ensemble** : Diviser l'ensemble d'entraînment en sous ensemble de manière aléatoire et avec répétition (c'est-à-dire que des exemples peuvent être répété dans différent sous-ensemble). Les différents sous ensemble peuvent être de même taille ou pas.
2. **Entraîner les arbres de décisions** : Entraîner un arbre de décision sur chaque sous-ensemble en utilisant l'algorithme décrit dans la section 4.2.3
3. **Prediction** : Pour l'ensemble de test, combiner les prédictions des arbres entraînés lors de l'étape 2. Traditionnellemnt on performe un vote entre les arbres de la forêt, c'est-à-dire que l'étiquette le plus prédite est choisi comme prédiction de la forêt mais on peut aussi appliquer n'importe quelle fonction de sommation, par exemple l'algorithme pourrait retourner la probabilité de chaque classe.

Une illustration de cet algorithme est montré dans la figure 4.3

4.3.2 Implémentation

Mon implémentation de la forêt aléatoire est conforme à l'agorithme 4.3.1 avec deux légères différences (DIARRA, 2024f). Tout d'abord, le critère utilisé par les arbres de décisions pour choisir le prédicateur le plus important est différent du gain d'information. Pour cela j'ai utilisé le critère de l'impurété **gini** qui est une autre mésure de l'incertitude d'une variable aléatoire définie par la formule :

$$Gini = 1 - \Sigma_i p_i^2$$

où p_i est la probabilité de la classe i. De manière similaire à l'entropie, une branche est pure si $Gini = 0$, ce qui implique que le probabilité d'une classe est à 1 et toutes les autres à 0. Mais contrairement à l'entropie dont la valeur maximale est 1, le score gini le plus mauvais est de 0.5, signifiant que toutes les classes ont la même probabilité et donc que la branche est parfaitement hétérogène, par exemple dans notre cas avec deux classes : $Gini_{max} = 1-(0.5^2+0.5^2) = 0.5$. Le prédicateur le plus important est donc choisi comme celui avec le plus petit score gini. Le score gini

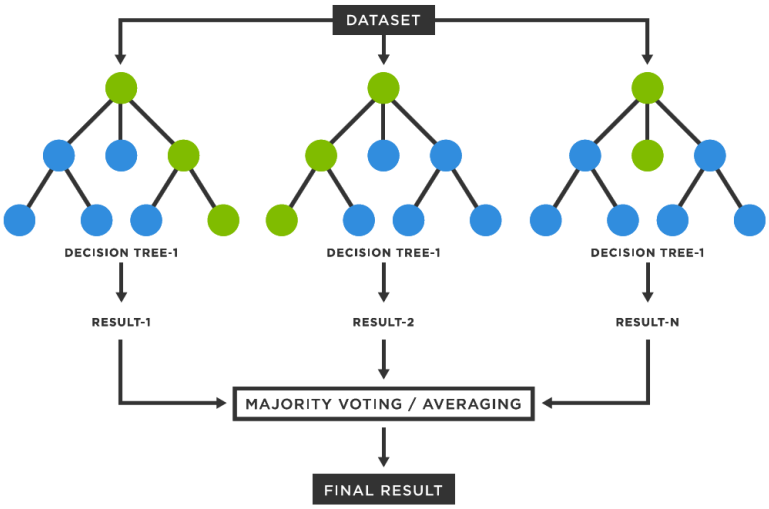

FIGURE 4.3 – Illustration de la forêt aléatoire (GUNAY, 2023)

a été choisi comme critère après avoir testé plusieurs mésures statistiques dont l'entropie et il s'est revélé être le plus rapide à calculer et donc le plus éfficace.

Ensuite, dans mon implémentation, les prédictions de la forêt sont retournées comme un vecteur représentant le probabilité de chacune des deux classes (fera défaut / ne fera pas défaut ; représentés respectivement par 1 et 0), la classification de la forêt pour chaque exemple des données de test est obtenu en appliquant un seuillage sur la probabilité de la classe positive (1) ; seuil que j'ai fixé à 0,055 soit (5,5%) pour maximiser la précision. Dans le chapitre 5 s'explique pourquoi ce seuil est si bas. La forêt est constituée de 200 arbres. Cet implémentation est basée sur celle de Scikit-learn (PEDREGOSA et al., 2011).

4.4 Le boost de gradient

Le boost de gradient est un autre algorithme d'ensemble mais appartenant à une autre famille d'ensemble que la forêt aléatoire : les boosteurs (**Boosting ensemble**). Le point commun reste la philosophie des ensembles [3] mais la méthode est différente.

En effet, là où les techniques de bagging comme la forêt aléatoire combine les prédictions de plusieurs apprenant faibles sur-ajustés pour avoir un modèle plus

3. Vulgairement dit, la philosophie des ensembles se base sur l'idée que en associant des apprenants faibles possédant divers bias et faiblesses on peut obtenir un modèle une approximation plus précise de la fonction qui génère les données

robuste et des prédictions corrigées par les divers bias des uns et des autres ; les techniques de boosting s'appuie sur la **superposition** de plusieurs apprenants faibles **sous-ajustés** [4], généralement aussi des arbres de décision, pour améliorer les performances globales de l'ensemble. L'idée de base est que chaque arbre tente de prédire et donc de corriger les erreurs de tous ses prédécesseurs (FRIEDMAN, 2001).

4.4.1 Algorithme

L'algorithme général est le suivant :

1. **Initialisation** : Initialiser le « réseau », avec un apprenant très faibles, généralement une constante comme la moyenne des étiquettes de l'ensemble d'entraînement ou les chances logarithmiques de chaque classe selon si le problème d'apprentissage est un problème de régression ou de classification.
   - **Régression** : Initialiser avec la moyenne des étiquettes (labels) de l'ensemble d'entraînement

     $$\hat{y}^{(0)} = mean(y)$$

   - **Classification** : Initialiser avec un modèle qui retourne la probabilité logarithmique [5] des différentes classes.

     $$\hat{y}^{(0)} = logit(y) = log(\frac{p}{1-p})$$

     où p est la probabilité de la classe positive dans l'ensemble d'entraînement si le problème est une classification binaire ou le vecteur des probabilités de chaque classe sinon.
2. **Iteration** : Calculez ensuite l'erreur résiduelle de ces premières prédictions pour entraîner un arbre avec celle-ci comme target. À part des changement arbitraire pour le critère de sélection de l'attribut le plus important, l'algorithme reste le même que celui décrit dans la section 4.2.3. Après l'arbre entraîné est ajouté à l'ensemble avec un taux d'apprentissage [6] en

4. Le sous-ajustement est l'inverse du sur-ajustement et se dit d'un modèle qui échoue à cerner les relations entre les variable indépendantes et la variable dépendante résultant en un modèle très instable.

5. La chance logarithmique d'une probabilité est le logarithme népérien de la probabilité que cet événement se produise. Il transforme une valeur de probabilité p allant de 0 à 1 en une valeur réelle allant de $-\infty$ à $+\infty$.

6. Le taux d'apprentissage est un nombre réel relativement petit utilisé comme une sorte de poids pour contrôler la vitesse d'apprentissage de l'ensemble. En gros il est là pour que la contribution d'un arbre à l'ensemble ne soit pas trop grand par rapport aux autres

le superposant aux arbres précédents où il servira à prédire l'erreur de ces derniers. Cette étape est répétée jusqu'à ce que nous atteignions un certain nombre d'arbres ou d'itération. Le processus d'ajout séquentiel d'un nouvel arbre pour prédire l'erreur du précédent est analogue à la descente de gradient dont on parlera dans la prochaine section, d'où le nom boost de **gradient**. Formellement ça donne :

Pour m de 1 à M (nombre d'iteration/arbre)

- **Calculez les erreurs résiduelle** : L'erreur résiduelle est la différence entre les vraies étiquettes et les prédictions de l'ensemble

$$r_i^{(m)} = y_i - \hat{y}_i^{(m-1)}$$

- **Entraîner un arbre de régression** : Entraîner un arbre de régression avec les données de l'ensemble d'entraînement comme entrées et les erreurs résiduelles comme sortie. Le but de cet arbre est de prédire l'erreur de l'ensemble, raison pour laquelle tous les arbres intermédiaires sont des arbres de régression.
- Ajoutez les prédictions du nouvel arbre pondérés par le taux d'apprentissage aux prédictions de l'ensemble actuel.

$$\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} + \eta \cdot h_m(x_i)$$

où $h_m(x_i)$ est la prédiction du nouvel arbre pour l'erreur de l'ensemble sur l'exemple i de l'ensemble d'entraînement.

3. **Prédiction final** : Enfin, le modèle fait une prédiction pour chaque x_i de l'ensemble de test en ajoutant les prédictions de tous les arbres (pondérées par le taux d'apprentissage). Pour un problème de classification, ce résultat est transmis à une fonction de seuillage ou de convertion en probabilité.
   - **Pour la régression** :

$$\hat{y}_i^{final} = \hat{y}^{(0)} + \Sigma_{m=1}^{M} \eta \cdot h_m(x_i)$$

   - **Pour la classification** : il suffit de reconvertir $\hat{y}_i^{final}$ qui sera sera alors la chance logarithmique des différentes classes grâce à une fonction comme celle ci (soyez attentif à cette fonction, on la reverra plus tard) :

$$p_i^{final} = \frac{1}{1 + e^{-\hat{y}_i^{final}}}$$

Une illustration de l'algorithme est montré dans la figure 4.4

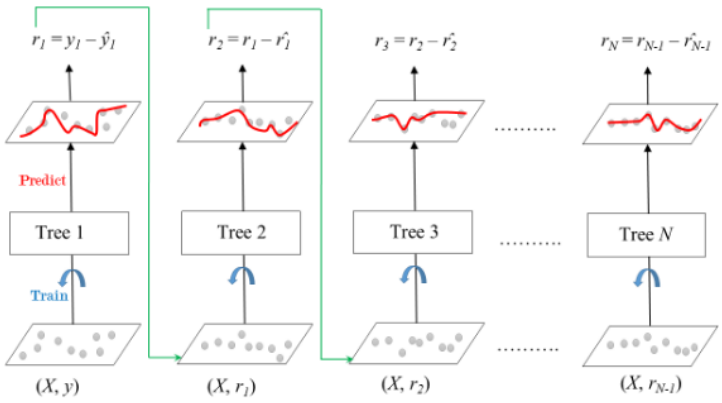

FIGURE 4.4 – Illustration de l'algorithme de boost de gradient (GEEKSFORGEEKS, 2023)

4.4.2 Implémentation

Le plus gros inconvénient des modèles non paramètriques en général est que le temps nécéssaire pour leur entraînement augmente avec la taille de l'ensemble d'entraînement et les techniques de boosting sont particulièrement démonstratifs de cette faille. Donc on utilise souvent des versions un peu modifiées, voir tronquées, de l'algorithme ci-dessus quand on a à faire à de grands ensembles de données (ODEGUA, 2020). Pour ce projet j'ai donc utilisé l'implémentation de scikit-learn basée sur les histogrammes qui est plus efficace grâce à deux particularités majeures :

- **Discrétisation des valeurs continues** : Les prédicateurs continues sont discrétisées en un nombre fixe de groupes au moment de la construction des arbres de décisions. Vous rappelez vous de l'exemple avec l'âge au début de la section 4.2 ? Là où les arbres de décisions classiques utiliseraient chaque valeurs de ces prédicateurs comme branche. Le fait de créer des catégories pour les valeurs continues (catégories qui seront les branches de cet attribut) réduit la précision des valeurs des attributs mais permet un calcul beaucoup plus rapide.
- **Créer des histogrammes** : Pour chaque prédicateur, un histogramme [7] est construit en comptant le nombre d'exemples dans chaque branche, d'où le nom **Histogram-based Gradient Boosting**. Ces histogrammes sont la représentation qui sera utiliser pour entraîner les arbres au lieu de l'ensemble complet, cela permet de reduire significantivement l'usage

7. Un histogramme est un graphique qui montre la fréquence des données numériques à l'aide de rectangles. La hauteur d'un rectangle (l'axe vertical) représente la fréquence de distribution d'une variable (le montant ou la fréquence d'apparition de cette variable).

mémoire et encore une fois d'accélérer l'entraînement des arbres.

Traditionnellement, le prédicateur le plus important (lors de l'entraînement des arbres) est choisi comme étant celui qui minimise la somme de la somme quadratique des erreurs résiduelles dans chaque branche. Cette somme est définie par la formule :

$$RSS = \Sigma_{n=1}^{M} Q_n$$

où Q_n est la somme quadradique des erreurs résiduelles dans la $n$ième branche définie par :

$$Q_n = \Sigma_i r_i^2$$

où r_i est l'erreur résiduelle du $i$ème exemple de la branche. En pratique, pour les problèmes de classifications les erreurs résiduelles sont souvent remplacées par les gradients d'une **fonction de coût** et le but est de minimiser cette dernière, on détaillera cette approche dans la prochaine section.

Tout comme l'implémentation de la forêt aléatoire mon implémentation du boost de gradient possède 200 arbres, le taux d'apprentissage est fixé à 0,1 et le nombre maximal de groupes lors de la discrétisation des valeurs continues est fixé à 255 ; notez que cette limite est aussi applicable aux prédicateurs de nature catégorique possédant plus de 255 valeurs uniques(Diarra, 2024e).

4.5 La régression linéaire

La régression linéaire est l'un des modèles paramétriques les plus anciens, peut-être même le modèle fondamental de cette philosophie. Elle a été introduite au début du XIXe siècle par le mathématicien et statisticien Francis Galton, qui l'a utilisée pour étudier la relation entre la taille des parents et celle de leur progéniture. Depuis lors, la régression linéaire a été largement étudiée et appliquée dans divers domaines, notamment l'économie et, bien sûr, l'apprentissage automatique.

Le but de la régression linéaire est de créer une approximation de la relation linéaire entre chaque variable indépendante et la variable dépendante, celà sous-entend donc que pour que la régression linéaire soit efficace les prédicateurs doivent décrire une certaine forme de linéarité avec la variable dépendante, c'est-à-dire une corrélation non nulle entre les variables d'entrées et la sortie. Lorsqu'il n'y a qu'une seule variable indépendante, on qualifie la tâche de : ***Régression linéaire simple ou univariée***. Et régression linéaire multi-variée s'il y a plus d'une variable d'entrée. Formellement, le problème de la régression linéaire est donc le suivant :

Étant donné un ensemble de données décrivant une corrélation linéaire entre des prédicateurs x_i et une variable de sortie (target) y ; trouver les valeurs des

paramètres w_i et b pour lesquels l'équation linéaire $y_j = b + \Sigma_i w_i \cdot x_{j,i}$ est vraie ou sensiblement vraie pour chaque exemple x_j dans l'ensemble d'entrainement. Les w_i sont appélés les **poids** et b le **bias** du modèle. Pour trouver ces valeurs, on cherche à minimiser une **fonction de coût**.

4.5.1 Fonction de coût

Une fonction de coût est un moyen de mésurer et de pondérer l'erreur d'un modèle, on l'appelle aussi fonction de perte. Une fonction de perte $L(y, \hat{y})$ est définie comme l'utilité perdue en prédisant $h(x) = \hat{y}$ lorsque la bonne réponse est $f(x) = y$. Une fonction de perte est différente d'un taux d'erreur, dans le sens où une fonction de perte ne s'intéresse pas seulement à l'erreur du modèle en prédisant $\hat{y}$ au lieu de y. Toute fonction qui capture la perte réelle résultant d'une mauvaise prédiction du modèle peut être une fonction de perte.

Prenons l'exemple de notre thème même, le risque encouru par l'institution prêteuse en accordant un prêt à une personne qui s'avère être un prêt défaillant est beaucoup plus grand que celui qu'elle encourt en refusant un prêt à quelqu'un qui aurait potentiellement pu le rembourser, raison pour laquelle certaines institutions durcissent leur critères sur les prêts et refusent beaucoup de demandes. Et pareil au niveau du client les risque ne sont pas les mêmes, donc un modèle qui prédit beaucoup trop souvent des défauts de paiement comme bon prêt causera plus de tort dans le monde réel qu'un autre modèle qui a le bias inverse. On peut faire une fonction de perte qui capture cette subtilité, c'est là la différence entre une fonction de coût et le taux d'erreur, par exemple on peut définir une fonction de coût de manière à ce que $L(y = \textit{remboursera}, \hat{y} = !\textit{remboursera}) = 1$ and $L(y = !\textit{remboursera}, \hat{y} = \textit{remboursera}) = 10$.

La fonction de coût est la mésure de performance de l'apprentissage supervisé car dans beaucoup d'algorithmes c'est elle que nous cherchons à minimiser. Pour expliquer les différents concepts impliqués dans l'entraînement de tels modèles dans le reste de ce document, nous allons nous baser sur une des formes les plus simples de fonction de coût, la fonction de perte $L2$ ou perte quadratique qui est définie comme :

$$L2(y, \hat{y}) = (y - \hat{y})^2$$

Lors d'un entraînement nous cherchons à minimiser la somme totale des pertes sur l'ensemble des données d'entraînement soit :

$$SumL2 = \Sigma_j (y_j - \hat{y}_j)^2$$

Dans le context de la régression linéaire, on le rappelle $\hat{y}_j = b + \Sigma_i w_i \cdot x_{j,i}$, ce qui donne comme perte :

$$SumL2 = \Sigma_j (y_j - (b + \Sigma_i w_i \cdot x_{j,i}))^2$$

Cette somme étant une équation avec $i + 1$ inconnues, pour chaque poids w_i et un bias b, elle atteint sa valeur minimale quand les dérivées partielles de la somme par rapport à chacune des inconnues sont égales à 0. C'est-à-dire lorsque :

$$\frac{\partial}{\partial w_i}\Sigma_j(y_j - (b + \Sigma_i w_i \cdot x_{j,i}))^2 = 0$$

pour chaque w_i de l'ensemble des poids et

$$\frac{\partial}{\partial b}\Sigma_j(y_j - (b + \Sigma_i w_i \cdot x_{j,i}))^2 = 0$$

Ces équations ont une seule solution unique (si elle existe), souvent celà implique que les valeurs idéales des poids et du bias sont des nombres réels avec plus de 50 décimales, leur calcul prend donc énormement de temps et on est souvent obligés de les tronquer de toute façon. Donc au lieu de chercher directement les solutions de ces équations on cherche à les approximer grâce à un algorithme qu'on appelle la **descente de gradient** qui est devenu la norme avec les modèles paramétriques. L'idée est que, par exemple, si les vraies valeurs de w_1 et b pour que leurs dérivées partielles respectives soient exactement égales à zéro sont : $w_1 = 1,702145614785401$ et $b = -0,8745123177559741$. Eh bien, nous serions assez satisfaits d'un algorithme qui convergerait vers $w_1 = 1,701$ et $b = -0,874$.

4.5.2 La Descente de gradient

Avec la descente de gradient, les valeurs des paramètres sont initialisées « aléatoirement » avec de petites valeurs, généralement issues d'une distribution gaussienne ou tout simplement mises à zéro au début de l'algorithme et nous visons à converger par itération vers une approximation des valeurs optimales des poids et du bias qui minimise la fonction de coût. L'algorithme est le suivant :

Pour un certain nombre d'époques [8] d'entraînement ou tant que l'algorithme n'a pas encore convergé (ce qui signifie qu'il y a encore des changements importants dans les valeurs des paramètres), on calculera les « gradients », c'est-à-dire les dérivées partielle de la fonction de perte pour chaque paramètres et chaque exemple d'entraînement x_j et on mettra à jour les valeurs de chaque w_i et de b suivant la direction de leurs gradients. Formellement ça donne l'algorithme suivant :

Pour m de 1 à M nombre d'époques

1. **Calculer les gradients** : Calculer les gradients de la fonction de perte par rapport à chaque paramètres.

8. Une époque d'entraînement réfère à une application de l'algorithme de descente de gradient sur l'ensemble d'entraînement entier.

— Pour un exemple d'entraînment x_j et un certain w_i, poids du $i$ème attribut de x_j on a donc :

$$\frac{\partial}{\partial w_i} L2 = \frac{\partial}{\partial w_i}(y_j - (b + w_i \cdot x_{j,i}))^2 = 2(y_j - (b + w_i \cdot x_{j,i})) \cdot \frac{\partial}{\partial w_i}(y_j - (b + w_i \cdot x_{j,i}))$$

$$= -2(y_j - (b + w_i \cdot x_{j,i})) \cdot x_{j,i}$$

— Pour le bias on a :

$$\frac{\partial}{\partial b} L2 = \frac{\partial}{\partial b}(y_j - (b + w_i \cdot x_{j,i}))^2 = 2(y_j - (b + w_i \cdot x_{j,i})) \cdot \frac{\partial}{\partial b}(y_j - (b + w_i \cdot x_{j,i}))$$

$$= -2(y_j - (b + w_i \cdot x_{j,i}))$$

Pour trouver les expressions ci-dessus il est juste nécéssaire de connaître les bases de la différentiation telles que : $\frac{\partial}{\partial w} wx = x$ et $\frac{\partial}{\partial w} w = 1$

2. **Mettre à jour les valeurs des paramètres** : Ensuite l'algorithme mettra à jour les valeurs des paramètres en utilisant la règle suivante :

— Pour chaque w_i :

$$w_i \leftarrow w_i - \eta \cdot \frac{\partial}{\partial w_i} L2$$

— Pour le bias :

$$b \leftarrow b - \eta \cdot \frac{\partial}{\partial b} L2$$

où η est le taux d'apprentissage, même principe que celui de la section 4.4.1

Une forme plus complète de l'algorithme est obtenu en sommant les mises à jours des poids pour chaque exemple d'entraînement x_j par rapport à la somme des pertes. Ce qui donne, pour j de 1 à N nombre d'exemples :

— Pour chaque w_i :

$$w_i \leftarrow w_i - \eta \cdot \Sigma_j \frac{\partial}{\partial w_i} SumL2 = w_i + 2\eta \cdot \Sigma_j^N (y_j - (b + w_i \cdot x_{j,i})) \cdot x_{j,i}$$

— Pour b :

$$b \leftarrow b - \eta \cdot \Sigma_j \frac{\partial}{\partial b} SumL2 = b + 2\eta \cdot \Sigma_j^N (y_j - (b + w_i \cdot x_{j,i}))$$

En pratique, au lieu de répeter ces opérations pour chaque prédicateurs (donc pour chaque w_i, ce qui va s'avérer très lent, on va plutôt vectorialiser toutes les équations précédentes depuis l'équation de prédiction du modèle. Ce qui va donner :

— L'équation du modèles devient :

$$\hat{Y} = b + W \cdot X$$

où W est la matrice des poids où $w_{j,i}$ est le poids associé au $i$ème attribut du $j$ème exemple d'entraînement, X est la matrice de tous les exemples d'entraînement avec tous les prédicateurs, cette matrice est de la même forme que W, le bias reste une valeur unique pour tout l'ensemble et $\hat{Y}$ est le vecteur des prédictions sur l'ensemble entier, de manière similaire Y est le vecteur des vraies étiquettes.

— La formule de mise à jour des poids par la descente de gradient devient :

$$W \leftarrow W - \eta \cdot \frac{\partial}{\partial W} SumL2 = W + 2\eta \cdot (Y - (b + W \cdot X)) \cdot X$$

où W sera une matrice dont on pourra prendre la moyenne ou la somme par colonne comme nouvelle valeur des w_i

— La formule de mise à jour du bias par la descente de gradient devient :

$$b \leftarrow b - \eta \cdot \frac{\partial}{\partial b} SumL2 = b + 2\eta \cdot (Y - (b + W \cdot X))$$

où b sera une matrice de bias dont on pourra ensuite prendre la moyenne ou la somme comme valeur unique pour b. Notez aussi que la plupart du temps on simplifie 2η à η tout simplement car le taux d'apprentissage est généralement un nombre très petit dont la multiplication par 2 reste semblable à ce dernier.

En répétant donc le processus ci-dessus pour M iteration sur l'ensemble de données entier, M époque, l'algorithme de descente de gradient va converger vers les valeurs de chaque paramètres pour lesquelles la somme des pertes est minimiser. Il est plus facile de raisonner avec les formes simples de ces équations qu'avec leur formes matricielles donc pour les prochaines démonstration où on aura besoin de revenir à ces équations j'utiliserai les formes simples. La figure 4.5 montre une illustration d'un modèle de régression linéaire univariée entraîner pour prédire le prix d'une maison avec une seule variable indépendante qui est sa superficie. On peut voir que le modèle, alors définie par juste un poids et un bias, décrit une droite qui correspond presque parfaitement aux données d'entraînement et qui peut être utilisé pour prédire une estimation du prix d'une nouvelle maison. L'idée pour les modèles multi-variée aussi est de trouver les valeurs des paramètres de l'hyperplan [9] qui décrit le même comportement pour un espace multi-dimensionnel.

9. En géométrie, un hyperplan est une généralisation d'un plan bidimensionnel dans un espace tridimensionnel à des espaces mathématiques de dimension arbitraire. Comme un plan dans l'espace, un hyperplan est une hypersurface plate, un sous-espace dont la dimension est inférieure d'une unité à celle de l'espace ambiant. Dans un espace bidimensionnel, c'est une ligne ; dans un espace tridimensionnel, c'est un plan, et ainsi de suite.

FIGURE 4.5 – Un modèle de régression linéaire simple décrit une droite qui suit la relation linéaire de l'ensemble de données

Tout comme pour les arbres de décisions je n'ai pas implémenté de modèle de régression linéaire pour ce projet. Comme son nom l'indique la régression linéaire est un algorithme de régression et la tâche de prédire les défauts de paiement est une tâche de classification, cependant expliquer la régression linéaire en profondeur comme ici me permettra d'être plus concis dans les explications des trois dernier modèles que j'ai testé.

4.6 La régression logistique

De par le fait qu'ils sont caractérisés par une ensemble fixe de paramètres (généralement égal au nombre de prédicateurs + 1 bias), les modèles de régression linéaire sont obligés de construire une hypothèse forte sur la fonction inconnue qui a génère les données. Parce que contrairement aux modèles non paramètriques comme les arbres de décisions, ils n'ont pas la capacité de changer de forme ou de grandir pour s'adapter parfaitement à l'ensemble d'entraînement, donc ils doivent faire des compromis pour trouver les meilleurs approximations des valeurs de poids et du bias qui minimise les pertes pour tous les exemples d'entraînement.

Cette particularité qui est censé être une contrainte de base s'avère finalement être la meilleure solution face au sur-ajustement. En effet ces modèles sont beaucoup moins sensible à ce problème, tout en faisant des prédictions correctes

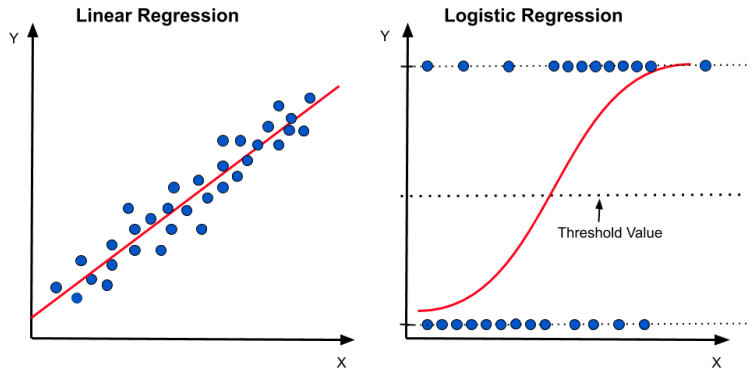

Figure 4.6 – Visualisation d'un modèle de régression logistique simple

la plupart du temps. On veut pouvoir mettre ce type de modèle au service des problèmes de classification également, il y'a donc deux possibilités majeures qui s'offre à nous : le **seuillage** ou la conversion de prédictions réelle continues en **probabilité**.

Avec la regression logistique on choisit la deuxième option afin de pouvoir faire de cette conversion une partie intégrante du modèle. Plus exactement, la régression logistique est une modification de la régression linéaire parfaitement pensée pour les problèmes de classification binaire (comme prédire les défaut de paiement). L'idée est de prendre un modèle de régression linéaire et de l'équiper d'une fonction qui puisse transformer la sortie $\hat{y}_j = b + \Sigma_i w_i \cdot x_{j,i}$ en la probabilité que l'exemple x_j appartienne à la classe positif. Cette fonction est connue sous le nom de fonction **sigmoid** et est définie par :

$$Sigmoid(z) = \frac{1}{1 + e^{-z}}$$

C'est bien la fonction de la fin de l'algorithme 4.4.1. Vous pouvez remarquez que quelque soit la valeur de z les valeurs de cette fonction sont dans l'intervalle $]0, 1[$.

En grosso modo, l'idée est de modifier légèrement un modèle de régression linéaire de manière à ce que les prédictions du modèle soit désormais sous la forme $Sigmoid(z)$ où $z = \hat{y}_j = b + \Sigma_i w_i \cdot x_{j,i}$ représentant la probabilité que l'exemple x_j appartienne à la classe positif (représentée par 1) qui dans notre cas signifie que le client fera défaut sur son prêt. La figure 4.6 montre l'effet de la fonction sigmoid sur les prédictions du modèle de régression et comment ces probabilités sont typiquement interprêtées.

4.6.1 Algorithme (Descente de gradient)

Le concept et l'algorithme de descente de gradient utilisé pour entraîner le modèle reste fondamentalement le même. Il y'a juste une petite nouvouté, puisque maintenant la prédiction du modèle n'est plus donnée directement par $\hat{y}_j = b + \Sigma_i w_i \cdot x_{j,i}$ mais plutôt par $Sigmoid(\hat{y}_j)$ on va avoir besoin d'une étape supplémentaire pour trouver le gradient de la fonction de perte par rapport aux différents poids et au bias parce que la fonction de coût $L2$ qu'on va encore utilisée pour l'illustration elle est toujours en fonction de $\hat{y}_j$ comme toute autre fonction de coût en soit. Ce qui nous amène au calcul suivant :

— Pour un exemple d'entraînment x_j et un certain w_i, poids du $i$ème attribut de x_j on a donc :

$$\frac{\partial}{\partial w_i} L2 = \frac{\partial L2}{\partial Sigmoid(\hat{y}_j)} \cdot \frac{\partial Sigmoid(\hat{y}_j)}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial w_i}$$

— Pour le bias on a :

$$\frac{\partial}{\partial b} L2 = \frac{\partial L2}{\partial Sigmoid(\hat{y}_j)} \cdot \frac{\partial Sigmoid(\hat{y}_j)}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial b}$$

Avec les démonstrations de la section 4.5.2 on sait que :

$$\frac{\partial L2}{\partial Sigmoid(\hat{y}_j)} = \frac{\partial}{\partial Sigmoid(\hat{y}_j)} (y_j - Sigmoid(\hat{y}_j))^2 = -2(y_j - Sigmoid(\hat{y}_j))$$

$$\frac{\partial \hat{y}_j}{\partial w_i} = \frac{\partial}{\partial w_i}(b + w_i \cdot x_{j,i}) = x_{j,i}$$

$$\frac{\partial \hat{y}_j}{\partial b} = \frac{\partial}{\partial b}(b + w_i \cdot x_{j,i}) = 1$$

Il ne reste donc plus qu'à dériver la fonction sigmoid par rapport à $\hat{y}_j$, on obtient donc :

$$(Sigmoid(\hat{y}_j))' = (\frac{1}{1 + e^{-\hat{y}_j}})' = \frac{-(-e^{-\hat{y}_j})}{(1 + e^{-\hat{y}_j})^2} = \frac{e^{-\hat{y}_j}}{(1 + e^{-\hat{y}_j})^2}$$

Au final on obtient :

— Pour un exemple d'entraînment x_j et un certain w_i, poids du $i$ème attribut de x_j :

$$\frac{\partial}{\partial w_i} L2 = -2(y_j - Sigmoid(\hat{y}_j)) \cdot \frac{e^{-\hat{y}_j}}{(1 + e^{-\hat{y}_j})^2} \cdot x_{j,i}$$

— Pour le bias :

$$\frac{\partial}{\partial b} L2 = -2(y_j - Sigmoid(\hat{y}_j)) \cdot \frac{e^{-\hat{y}_j}}{(1 + e^{-\hat{y}_j})^2}$$

L'algorithme de descente de gradient reste exactement le même et les mêmes concepts s'applique. Ce qui est très intéressant avec les nouveaux modèles et démonstrations qu'on voit depuis le section 4.5 c'est qu'on peut remarquer que ces concepts et algorithmes peuvent se généraliser à d'autres fonctions et problèmes. Donc il est techniquement possible de remplacer la fonction *L2* et la fonction *Sigmoid* avec n'importe quelle autre fonction différentiable. Cela a permit le développement de d'autres algorithmes comme les réseaux de neuronnes artificiels.

4.6.2 Implémentation

Mon implémentation de la régression logistique est assez « peu conventionnelle » on va dire, puisque je l'ai implémenté avec keras qui est un framework (une structure) pour l'apprentissage profond normalement (CHOLLET et al., 2015). Cela a donné un modèle avec des particularités qu'on associe typiquement aux réseaux de neuronnes, par exemple la descente de gradient est éffectuée par l'algorithme du **Moment adaptatif (AdaM)** qui est une version légèrement modifiée de l'algorithme de descente de gradient ordianaire décrit dans la section 4.5.2 ; de plus la mise à jour des paramètres se fait par lot, c'est-à-dire que les gradients sont accumulés et la descente se fait pour un lot à la fois, qui est une matrice de M exemples d'entraînement sur N au total au lieu de le faire pour chaque exemple individuellement ou pour une grande matrice de tout l'ensemble comme je l'avais précédemment évoqué. Ceci étant dit le fonctionnement du modèle est conceptuellement identique à celui d'un modèle de regression logistique et peut donc être considéré comme tel.

La raison pour laquelle j'ai décidé d'implémenter la régression logistique comme ceci est de faciliter la transition vers les réseaux de neuronnes dans ce document et pour ceux qui liront les codes.

Mon implémentation utilise aussi une fonction de coût différente de *L2* qui est pensée particulièrement pour les problèmes de classification et classification binaire notamment. C'est la fonction de **cross-entropie binaire** ou log-loss définie par :

$$LogLoss = -[y_j \cdot log(p_j) + (1 - y_j) \cdot log(1 - p_j)]$$

où $p_j = Sigmoid(\hat{y}_j)$ et donc la forme complète est la somme normalisée des pertes :

$$LogLoss = -\frac{1}{N} \Sigma_j^N [y_j \cdot log(p_j) + (1 - y_j) \cdot log(1 - p_j)]$$

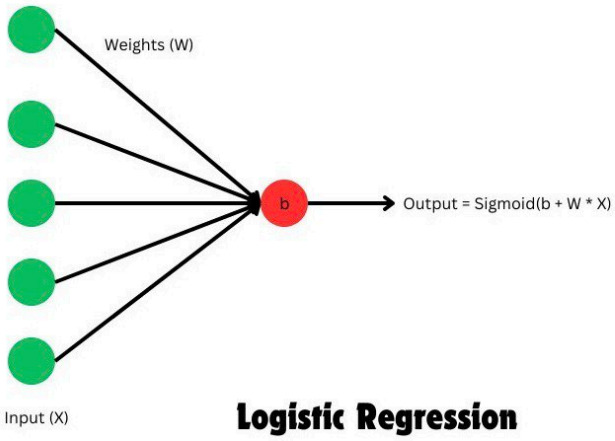

FIGURE 4.7 – Un modèle de régression logistique vu comme un neuronne

où N est le nombre d'exemple. De par son expression cette fonction de perte atteint sa valeur maximal quand y_j est trop différent de p_j, une manière de pénaliser plus lourdement le modèle quand il a gravement tort et donc d'avoir des gradients plus grands pour rémedier à ça.

Le modèle a donc été entraîner sur ces bases là pour sept (07) époques sur des lots de 64 exemples et avec un taux d'apprentissage de 0,001 (DIARRA, 2024f).

4.7 Les réseaux de neurones artificiels

Revenons au modèle de régression logistique de la section 4.6, maintenant au lieu de le voir comme un modèle à part entière je vous suggère de penser à ce modèle comme une simple unité de calcul, une petite partie d'un modèle beaucoup plus grand. C'est assez peu commun d'expliquer les résaux neuronaux de cette manière mais c'est bien là les fondements de cet algorithme.

Les réseaux de neuronnes artificiels sont un type de modèle paramétriques qui marche grâce à la superposition de plusieurs couches de petites unités de calcul qui sont traversées une à une par les différents attributs $x_{j,i}$ d'un exemple de donnée x_j. Les sorties d'une couche devenant les entrées de la suivante jusqu'à une couche de sortie représentant la prédiction du modèle pour l'exemple x_j (cette couche peut aussi contenir plusieurs unités de calculs). Ce sont les fondements de toute une famille d'algorithmes basés sur ce concept et d'un sous

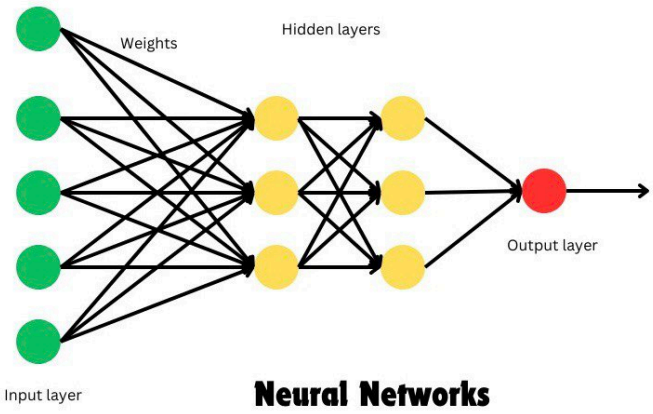

FIGURE 4.8 – Un réseau neuronal à action directe vu comme un ensemble de modèle de régression logistique

domaine de l'apprentissage automatique appélé : **Apprentissage profond** [10].

Les unités de calcul vont être appélées « neuronnes » et le type de modèle que je décris dans le paragraphe précédent est le type de modèle d'apprentissage profond le plus simple appélé : réseau neuronal à action directe ou **feedforward neural network**. Comme je l'ai dit dans la section 4.6.2, l'implémentation que j'ai fait de la régression logistique était faite ainsi dans le but de faciliter l'analogie et la transition aux réseaux de neuronnes. En effet si l'on considère un modèle de régression logistique comme un seul neuronne unique comme l'illustre la figure 4.7, il dévient beaucoup plus simple de raisonner avec les réseaux neuronaux à action directe, vous pouvez les voir comme des modèles qui superpose plusieurs modèles de régression logistique (neuronnes) par couche. La figure 4.8 montre un réseau neuronal directe avec 2 couches intermédiaires ou couches cachées *(hidden layers)* et un seul neuronne de sortie pensé pour la classification binaire.

Ajoutez à celà le fait qu'on a la possibilité de remplacer la fonction sigmoid par toute autre fonction pour chaque neuronne en théorie (même si en pratique et par covention les neuronnes de la même couche utilisent la même fonction) et on a toutes les bases de l'apprentissage profond. Pour un neuronne la fonction

10. L'apprentissage profond est un sous domaine de l'apprentissage qui utilise des réseaux neuronaux artificiels et d'autres algorithmes compexes pour simuler le pouvoir de décision complexe du cerveau humain. Ce genre de modèle alimente la majeure partie de l'intelligence artificielle aujourd'hui.

qui permet de passer de $\hat{y}_j = b + \Sigma_i w_i \cdot x_{j,i}$ à l'entrée du prochain neuronne est ce qu'on appelle une **fonction d'activation**. Pour simplifier les démonstrations dans ce document on va supposer que toutes les fonctions d'activations sont la fonction sigmoid.

4.7.1 La rétro-propagation

La rétro-propagation est une variante de l'algorithme de descente de gradient qui vise à adapter cette dernière à l'entraînement de réseaux de neuronnes multi-couches (NIELSEN, 2018). L'algorithme est presque identique à la descente de gradient, juste qu'il apporte un support pour calculer les gradients des poids et des bias qui n'appartienne pas au neuronne de la couche de sortie en propageant l'erreur de la dernière couche aux précédentes grâce à ses gradients, d'où le non **rétro-propagation**. L'algorithme de rétro-propagation appliqué au poids et au bias du neuronne de sortie qui prédit la probabilité qu'un exemple x_j appartienne à la classe positive revient à la descente de gradient sur un modèle de régression logistique.

Avez vous pris soins de remarquer cette expression dans la section 4.6.2 ?

$$\frac{\partial}{\partial w_i} L2 = \frac{\partial L2}{\partial Sigmoid(\hat{y}_j)} \cdot \frac{\partial Sigmoid(\hat{y}_j)}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial w_i}$$

Cette décomposition est connu sous le nom de : **règle de la chaîne des gradients** et elle est le fondement de l'algorithme de rétro-propagation car en interpolant ainsi des termes qui font le lien entre la fonction de perte et un poids ou un bias donné on peut trouver les expressions des gradients de fonction de perte par rapport à n'importe quel poids ou bias du réseau neuronal. L'algortihme de rétro-propagation appliqué aux couches cachées du réseau revient donc simplement à suivre la chaîne. Ce qui est en revanche très fastidieux puisque la chaîne s'aggrandit au fur et à mésure qu'on remonte dans le réseau.

Tout cela paraît assez complexe pour quelqu'un qui n'est pas particulièrement bon en math et croyez moi j'ai souffert pour en arriver à ce petit niveau de compréhension. En résumé, la seule réelle différence entre l'algorithme de rétro-propagation et la descente de gradient est la taille de la chaîne des gradients. Les autres étapes une fois qu'on a les gradients de la fonction de perte par rapport à tous les poids et tous les bias restent identiques à la descente de gradient vu dans la section 4.5.2.

4.7.2 Implémentation

Comme dit précédemment, j'ai fait une implémentation avec keras d'un réseau de neuronnes à action directe. Après plusieurs tests, l'architecture qui conci-

liait les performances et le temps d'entraînement du modèle est composée d'une seule couche cachée de 100 neuronnes et d'une couche de sortie avec un seul neuronne. Les neuronnes de la couche cachée ont comme fonction d'activation la fonction d'unité linéaire rectifiée (**relu**) définie comme :

$$relu(z) = max(0, z)$$

En plus d'introduire une forme de non linéarité dans le fonctionnement du modèle, la fonction d'unité linéaire rectifiée et ses variantes sont généralement préférées pour les couches cachées en raison de la simplicité de leurs gradients ce qui permet d'accélerer considérablement la descente de gradient.

Pour l'unique neuronne de sortie, cela va de soit, la fonction sigmoid s'impose pour que les prédictions finales du modèle puisse être interprêtées comme la probabilité d'un exemple donnée de faire défaut sur son prêt. Et puisque le problème reste une classification binaire la fonction de coût reste la même que pour la régression logistique à savoir **log-loss**. Le meilleur modèle a été entraîner pour 10 époques d'entraînement sur des lots de 64 exemples par descente de gradient et avec un taux d'apprentissage de 0,001 encore une fois (DIARRA, 2024f).

4.8 Les machines à vecteur de support

Les machines à vecteur de support (**SVMs**) sont une famille d'algorithmes de classification dont l'objectif principal est de trouver une hyperplan qui sépare les exemples des différentes classes de manière optimale. Pour cela les SVMs cherchent à maximiser la marge, c'est-à-dire la distance entre l'hyperplan et les points de données (exemples) les plus proches de chaque classe. Ces points les plus proches sont appelés **vecteurs de support**. L'algorithme général peut être décrit comme suit :

1. Quand l'ensemble de données est directement linéairement séparable, les SVMs cherchent à trouver les bonnes valeurs des paramètres w_i et b de l'hyperplan

   $$b + \Sigma_i w_i \cdot x_i = 0$$

   qui maximise la marge. Les x_i sont les attributs ou prédicateurs de l'ensemble de données. Ceci est formulé comme un problème d'optimisation où la fonction à minimiser est :

   $$\frac{1}{2}\Sigma_i ||w_i||^2$$

   sous contraintes

   $$y_j(b + \Sigma_i w_i \cdot x_{j,i}) \geq 1$$

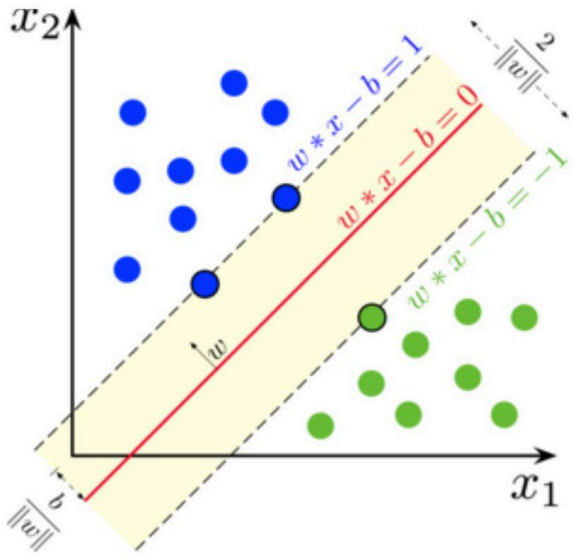

FIGURE 4.9 – Machines à vecteur de support pour une classification binaire

pour tout exemple x_j.

2. Quand l'ensemble de données n'est pas directement linéairement séparable, les SVMs utilisent des fonctions pour transformer les données dans un espace dimensionnel plus grand où elles peuvent être séparées linéairement.

3. Pour les ensembles de données non linéairement séparables, les SVMs introduisent des variables de relâchement ζ_i pour permettre certaines violations de la marge. Le problème de minimisation dévient donc :

$$min(\frac{1}{2}\Sigma_i||w_i||^2 + \epsilon \cdot \zeta_i)$$

où ϵ est un paramètre qui contrôle le compromis entre maximiser la marge et minimiser les erreurs de classification. Encore une fois sous les contraintes :

$$y_j(b + \Sigma_i w_i \cdot x_{j,i}) \geq 1 - \zeta_i; et \zeta_i \geq 0$$

La figure 4.9 montre une illustration de l'algorithme général appliqué à des données linéairement séparable pour une classification binaire. En pratique une approximation des valeurs des w_i et b grâce à la descente de gradient est toujours possible et toujours préférable avec les grands ensembles de données.

4.8.1 Les machines à vecteur de support pour la détection d'anomalie

Un détail que je n'ai pas précisé avec les autres modèles est que l'ensemble de données que j'avais à disposition pour ce travail était très déséquilibré. Sur les 1.526.659 exemples il n'y avait que 47.994 qui avait faillit sur leur prêt, soit approximativement 3% de l'ensemble de données. C'est une situation qui peut arriver dans certaines applications où les différentes classes sont naturellement déséquilibrés, la section 4.9 sera dédiée à expliquer ce problème et les solutions que j'ai mis en place pour essayer de minimiser l'impact sur les modèles de classification.

Cette section n'était pas censé faire partie du document initialement, tout vient de l'idée qu'avec un ensemble de données si déséquilibré, un modèle de detection d'anomalies [11] pourrait s'averer meilleur pour distinguer les deux classes. Donc l'idée m'a été proposer d'implémenter un modèle de detection d'anomalies qui sera utiliser comme un modèle de classification dans le soucis de déterminer correctement les clients qui sont plus susceptibles de faire défaut sur leur prêt.

Dans un papier de 1999, Bernhard Scholkopf et al ont proposé un algorithme de machine à vecteur de support pour aborder le problème de détection d'anomalies en essayant d'estimer un fonction f qui est positive sur un sous-ensemble S des données (considéré comme normal) et négative sur le complément (considéré comme anomalie). Leur méthode est un algorithme d'apprentissage non supervisé qui prend un ensemble de données X le projette dans un espace dimensionnel plus grand et cherche les paramètres de l'hyperplan qui englobe la majeur partie des données tout en maximisant la marge entre les points de données et l'origine de l'espace dimensionnel (SCHÖLKOPF et al., 1999). Le resultat est un modèle qui, une fois entraîné, évalue si un nouveau point de donnée tombe dans cet hyperplan (normal) ou à l'extérieur (anomalie).

4.8.2 Impémentation

J'ai donc testé cet algorithme, en entraînant un modèle sur un sous-ensemble des clients qui avait remboursé leurs prêts (un peu à la dernière minute), ce qui a justifié l'ajout de cette section. L'implémentation est basé sur celle de scikit-learn de l'algorithme de Scholkopf et al. Par contre l'algorithme est très inefficace en termes de temps ce qui m'a obligé à entraîner seulement sur 100.000 exemples (DIARRA, 2024a).

11. La détection d'anomalies est un domaine essentiel de l'analyse de données et de l'apprentissage automatique, qui consiste à identifier des observations ou des événements rares et atypiques dans un ensemble de données. Ces anomalies peuvent indiquer des erreurs, des fraudes, des pannes ou d'autres phénomènes inhabituels qui nécessitent une attention particulière.

4.9 Gérer le déséquilibre des données

Comme je l'ai évoqué dans la section 4.8.1, le déséquilibre des données est un aspects pas si rare que cela avec les ensemble de données. Enfaîte certains ensembles de données le sont ne serait-ce que de par la nature de la tâche que l'on cherche à faire avec, par exemple la detection de tumeurs par images. Dans cette application de la vision par ordinateur, l'ensemble de données est typiquement constitué d'images d'échographies (entre autres) où chaque pixel est étiquetté comme cancereux ou non selon la zone de l'image qui est atteint d'une tumeur sur le corps du patient. Dans un dataset pareil, des millions de pixels vont être étiquetté comme "normal" là où à peine un millier portera l'étiquette inverse et heureusement d'ailleurs, on ne souhaite pas que les gens ait des cancers énormes.

On parle donc d'ensemble de données déséquilibré quand une étiquette en particulier est très largement en supériorité numérique par rapport à l'autre ou aux autres. Ce déséquilibre en lui même apporte des informations, que ce soit sur la manière dont les données ont été collectées, le lieu et même des informations sur la fonction qui génère les données.

Cependant ce déséquilibre pose un problème pour l'apprentissage des modèles, du fait de la rareté de la classe minoritaire, les algorithmes que nous avons vu de la section 4.2 à 4.7, qui sont des algorithmes de classification vont vite même très vite converger vers le minimum de leur fonction de perte sans pour autant que le modèle résultant n'est appris quoi que ce soit des données. Les données fournies par Home Credit Group contenait des informations sur plus d'un million et demi de demandes de prêt parmi lesquelles juste 3% ont échoué à rembourser le prêt accordé. D'un côté on est bien content pour eux parce que ça veut dire que Home Credit Group et leur clients gèrent très bien les crédits mais de l'autre ça complique la tâche de l'ingénieur en apprentissage automatique de dévélopper un modèle qui puisse réellement distinguer les bons des mauvais crédits.

Vous l'avez surêment déviné, le problème avec les ensembles de données déséquilibrés est que ils exploitent le plus gros hack [12] des algorithmes d'apprentissage supervisé : **la fonction de perte**. En éffet pour qu'une fonction de perte d'un problème d'apprentissage supervisé puisse permettre d'extraire des connaissances des données dans un problème de classification, les différentes classes sont assumées d'avoir la même probabilité dans l'ensemble de données de sorte à ce que le pire qu'un modèle puisse avoir en termes de justesse de classification en prédisant toujours la même classe soit égale à cette probabilité.

Pour prendre l'exemple sur notre problème de classification binaire, la fonction log-loss est pensée de sorte à ce qu'un modèle qui prédit toujours la même

12. Un hack est une stratégie construite sur une faille d'un système afin de détourner ce dernier ce son fonctionnement normal ou de produire d'autres éffets qui n'avait pas été anticipés par le créateur du système. Le mot vient du verbe anglais **to hack** qui veut dire forcer un système.

classe (disons la classe 0 par exemple) est une perte totale extrêmement grande pour l'ensemble (en supposant que ce dernier est équilibré). Vous vous rappelez de l'expression de la fonction log-loss à la fin de le section 4.6.2 ?

Un modèle qui prédit toujours la classe 0, disant donc que le client remboursera son prêt, a une probabilité de prédiction de 1 (100%) pour la classe 0 et de 0 (0%) pour la classe 1. Ce qui donne une perte égale à 0 pour la classe 0 et ∞ pour la classe 1. En théorie $log(0)$ n'est pas définie et l'infini n'est pas un nombre, mais dans notre context ici on aura pas à ce préoccuper de ça car comme je l'ai dit p_j dans la formule de log-loss est la prediction du modèle qui est donnée uniquement comme la probabilité que le l'exemple x_j appartienne à la classe 1. Et cette probabilité est donné par la fonction sigmoid qui elle ne varie que dans l'intervalle ouvert]0,1[(signifiant que 0 et 1 sont exclus). Donc en pratique un modèle qui prédit tout le temps la classe 0 est un modèle qui donne toujours une très faible probabilité de la classe 1, pour simplifier ici va choisir $p_j = 0,0001$. Donc pour un modèle de classification qui prédit tout le temps la classe 0 avec $p_j = 0,0001$, en supposant un ensemble de données équilibré de N exemples, on a :

— Pour tous les exemples de la classe 0 :

$$LogLoss = -[0 \cdot log(0,0001) + (1-0) \cdot log(1-0,0001)] = 0,0001 \approx 0$$

— Pour tous les exemples de la classe 1 : :

$$LogLoss = -[1 \cdot log(0,0001) + (1-1) \cdot log(1-0,0001)] = 9,21$$

— Pour l'ensemble :

$$LogLoss = \frac{1}{N}(\frac{N}{2} \cdot 0 + \frac{N}{2} \cdot 9,21) = \frac{1}{N}(\frac{N}{2} \cdot 9,21) = \frac{9,21}{2} \approx 4,605$$

Pour une fonction dont le minimun est autour de zéro on est bien loin, cela implique que ce modèle aura raison 50% du temps en prédisant la même classe puisque l'ensemble est équilibré mais une perte totale de presque 5 et on le rappelle le but est de minimiser la fonction de perte ce qui conduira l'apprentissage.

En révanche le même modèle avec un ensemble déséquilibré sera dans un confort énorme qui ne permettra pas d'apprentissage. En supposant maintenant qu'on a un ensemble avec 97% de classe 0 comme avec celui de Home Credit Group. La perte totale du modèle sera approximativement :

$$LogLoss = \frac{1}{N}(\frac{9,7N}{10} \cdot 0 + \frac{0,3N}{10} \cdot 9,21) = \frac{1}{N}(\frac{0,3N}{10} \cdot 9,21) = \frac{0,3 \times 9,21}{10} \approx 0,27$$

C'est la même fonction de coût, le même modèle en termes de compétences mais deux réalités très différentes. Dans le second cas le modèle sera dans une

sorte de confort qu'il n'a pas dans le premier cas, car l'algorithme lui fera comprendre que les pertes sont déjà proches du minimum alors que le modèle n'a pas plus de pouvoir de prédiction que si l'on jouait la décision à pile ou face dans le fond. Le modèle comprend donc qu'en ne faisant aucun éffort dans ce nouvel environnement il a quand même raison 97% du temps et les pertes sont minimisées.

Cette situation est la raison pour laquelle j'ai décidé d'implémenter l'algorithme décrit dans la section 4.8.1 en tant que système de classification. Car si tous les algorithmes supervisés de classification sont sujets au problème décrit ci-haut avec les ensemble de données déséquilibrés (car ils ont tous dans le fond le même objectif). Un algorithme de détection d'anomalies non supervisé, de par sa nature même n'y est pas confronté et sera donc (en théorie) plus capable de reconnaître les exemples de la classe 1 (positifs). La section 5.2 fournit une évaluation comparative des différents modèles. Mais avant donc d'en arriver à détourner un algorithme de detection d'anomalies pour faire de la classification, j'ai essayé certains techniques pour tenter de minimiser l'impact du déséquilibre sur les modèles de classification.

4.9.1 Sur et sous échantillonnage

Le **sous-échantillonnage** est une technique conçue pour éssayer de gérer les ensembles de données déséquilibrés lors de l'entraînement de modèle de classification. Différentes implémentations existent mais l'idée de base est de supprimer certains exemples de la classe majoritaire jusqu'à atteindre une certaine forme d'équilibre. Ces exemples peuvent être supprimer soit aléatoirement (au risque de perdre des exemples intéressants) soit selon quelques critères comme par exemple supprimer les exemples de la classe majoritaire qui sont trop proches d'un ou plusieurs exemples de la classe minoritaire de par les valeurs de leur attributs.

Le **sur-échantillonnage** est la technique inverse, dans sa forme la plus simple elle consiste juste à répéter certains exemples de la classe minoritaire dans l'ensemble de données de manière aléatoire jusqu'à atteindre l'équilibre. D'autres variantes permettent de générer des exemples de la classe minoritaire en fonction de ceux qui sont déjà présents, dans cette catégorie le plus connu est sans doute **SMOTE** ou la technique de suréchantillonnage minoritaire synthétique de son nom complet en français. Avec SMOTE l'idée est de générer synthétiquement un exemple qu'on attribut à la classe minoritaire en faisant la différence entres les valeurs des prédicateurs d'un exemple de la classe minoritaire et un ou plusieurs de ses plus proches voisins dans l'espace des prédicateurs (en fonction du nombre d'exemple qu'on veut générer) et de multiplier toutes les valeurs obtenues par un nombre réelle choisit au hasard entre 0 et 1. Les valeurs obtenues à

la fin seront les valeurs des prédicateurs d'un nouvel exemple qui sera attribuer à la classe minoritaire (CHAWLA et al., 2002).

Les deux familles de techniques permettent de passer d'un ensemble de données déséquilibré à un autre qu'on a donc forcé à être équilibré, mettant les algorithmes de classification dans leur zone de confort pour qu'ils puissent apprendre à distinguer les différentes classes. En revanche, pour un ensemble de données qui par nature est déséquilibré, c'est-à-dire que le déséquilibre des données ne vient pas de la méthode de collecte mais d'une réalité du terrain, l'utilisation des techniques de sur et sous ajustements présente le risque d'entraîner et de valider un modèle qui va au final drastiquement chuter en performances quand il sera de nouveau confronté à un ensemble de données déséquilibré. Particulièrement si les prédicateurs n'ont pas été assez travaillés ou qu'ils entretiennent des rélations complexes avec la variable dépendante, dans ce genre de cas équilibrer les données peut donc devenir l'arbre qui cache la forêt.

J'ai été confronté à ce problème en utilisant ces techniques, j'ai donc décidé de ne pas tenter d'équilibrer l'ensemble de données et d'essayer de construire des modèles de classification correctes et conforme à la réalité de l'ensemble déséquilibré. Pour cela j'ai adopté une autre technique mathématiquement beaucoup plus fiable.

4.9.2 Pondérer les classes

La technique que j'ai finalement utilisé pour essayer de minimiser l'impact du déséquilibre des données sur les modèles de classification attaque directement la source du problème à savoir la fonction de coût. L'idée est de fixer un poids sur les erreurs du modèles de sorte à ce que l'erreur de classifier un exemple de la classe 1 soit beaucoup plus grande que l'erreur inverse. Si vous vous rappelez bien j'avais parlé de cela en introduisant les fonctions de perte dans la section 4.5.1. L'idée est que les erreurs du modèle ne contribuent pas équitablement à la fonction de perte et donc ainsi de forcer l'algorithme à faire plus attention à la classe dont la mauvaise classification est la plus coûteuse. Mathématiquement il s'agit juste de multiplier le résultat de la fonction de perte par un facteur différent pour chaque classe qui représente le poids d'une erreur sur la classification d'un exemple de cette classe :

$$LogLoss = \lambda_i \cdot LogLoss$$

pour chaque classe i de l'ensemble de données λ_i est son poids. Dans notre cas, ces poids ont été calculés en fonction de la proportion d'exemple dans chacune des deux classes, donnant un poids de sensiblement 0.516 pour la classe 0 (remboursera) et 15.61 pour la classe 1 (ne remboursera pas).

Chapitre 5

Évaluation

Dans ce chapitre nous portons l'attention sur l'évaluation des modèles que j'ai entrainé et donc indirectement des algorithmes qui sont derrières. Dans les deux prochaines sections j'explique les différents critères que j'ai choisi pour juger les modèles, comment ces derniers peuvent être interprêtés et qu'est ce qu'ils nous disent sur les modèles et les données.

5.1 Métriques

Un métrique est tout simplement une unité de mésure, dans le contexte de l'apprentissage automatique et des problèmes de classification il réfère surtout à une statistique qui résume en un nombre la capacité d'un modèle à distinguer les différentes classes. Alors techniquement on pourrait utiliser la fonction de perte comme un métrique mais il faudra aussi expliquer tout ce qui lui est liée pour que les gens puissent correctement interprêter ses valeurs. Pour cette raison, on choisit généralement des métriques plus simples pour évaluer un modèle.

Quand on a un modèle de classification binaire déjà entraîné, on cherche à attester le pouvoir de prédiction du modèle afin de savoir quel niveau de confiance peut-on placer en lui. Pour cela on va demander au modèle de prédire les classes de certains exemples qui ne se trouvaient pas dans l'ensemble d'entraînement et donc pour lesquels le modèle ne connaît pas la vraie éttiquette mais nous si. Ce nouvel ensemble de données étiquettées est appélé : **ensemble de test** et il est typiquement enlevé de l'ensemble des données avant l'entrainement créant un ensemble de données pour l'entraînement et un autre ensemble (généralement beaucoup plus petit) pour tester les modèles entraînés. Comme je l'ai dit à la fin de la section 3.3, dans mes traveaux l'ensemble de test correspond à **5%** de l'ensemble de données soit à peu près 77.000 exemples sur 1.526.659. Les statistiques que nous allons analyser dans la prochaine section sont donc les résultats

	Prédicion positive	Prédicion négative
Étiquette Positive	VP	FN
Étiquette Négative	FP	VN

TABLE 5.1 – La matrice de confusion

des différents modèles sur cet ensemble de test, sauf pour le modèle de détection d'anomalies qui n'a été entraîné que sur les exemples de la classe 0 et à donc un ensemble de test un peu plus grand (environ 55.000 de plus) constitué d'exemples des deux classes.

Pour revenir dans le vif du sujet, quand nous avons un modèle déjà entraîné que nous souhaitons utiliser pour de la classification, il n'y a que quatre cas de figure possibles : Soit le modèle prédit à raison qu'un exemple x_j est de la classe positive ($y_j = \hat{y}_j = 1$) où y_j est la vraie étiquette et $\hat{y}_j$ la prédiction du modèle, on va appeler ce genre de prédiction correcte **VRAI POSITIF (VP)** ; soit le modèle prédit à raison que l'exemple est de la classe négative ($y_j = \hat{y}_j = 0$), ce qu'on va appeler **VRAI NÉGATIF (VN)** ; soit il prédit à tort que l'exemple est de la classe positive ($y_j = 0$ et $\hat{y}_j = 1$) **FAUX POSITIF (FP)** ; soit enfin il prédit à tort que l'exemple est de la classe négative ($y_j = 1$ et $\hat{y}_j = 0$) **FAUX NÉGATIF (FN)**. De là on peut tirer le tableau 5.1 appélé **Matrice de confusion**.

Traditionnellement le métrique par excellence des modèles de classification est la **justesse de classification (Accuracy)** qui mésure le pourcentage de prédiction correcte que le modèle a fait :

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

Le problème est que ce métrique ne convient pas du tout à un ensemble de données déséquilibré car comme on la vue précédemment, un modèle qui prédit la classe 0 à chaque fois sera juste dans 97% des cas avec l'ensemble de données que nous avons. Ce qui fait que la justesse de classification uniquement ne nous dira rien sur le niveau réelle du modèle à part qu'il triche.

Par conséquent, on a besoin de métriques plus adaptés qui nous disent exactement ce que l'on veut savoir sur le modèle. Pour ma part j'ai choisi de me concentrer sur des métriques qui nous disent plutôt à quel point le modèle est capable de reconnaître les exemples de la classe positive, c'est-à-dire à quel point le modèle est bon pour reconnaître les clients qui feront défaut sur leur prêt. Selon moi, c'est la capacité la plus crucial dans notre cas précis, un modèle beaucoup trop tolérent avec la classe positive causera de serieux dégats si il était utiliser par des professionnels dans le monde réel.

Le premier métrique que j'ai rajouté à la justesse de classification est le **rappel (recall)** qui mésure le pourcentage d'exemple de la classe positive que le

modèle a correctement détecté, il est défini par :

$$recall = \frac{VP}{VP + FN}$$

Ce métrique nous permettra de tester la capacité du modèle à reconnaître les clients qui sont plus susceptibles de faire défaut sur leur prêt.

Le second métrique que j'ai donc ajouté à la justesse de classification et au rappel est la **précision** qui mésure le pourcentage d'exemples prédit comme positifs par le modèle et qui étaient vraiment positifs. Ce métrique est très important car il permet de mésurer l'effet des poids des différentes classes sur le modèle car si l'on pénalise gravement les erreurs de classification de la classe positive comme j'ai dit l'avoir fait dans la section 4.9.2 ; il y'a des chances que le modèle se mette à crier beaucoup trop souvent au loup, c'est-à-dire prédire la classe positive au moindre soupçon, ce qui aura certes pour effet d'augmenter le pourcentage de la classe positive qu'il detectera correctement (augmenter le rappel) mais au prix de classifier injustement plus de bonnes démandes potentielles comme des gens qui échoueront à rembourser leur prêt. La précision nous permettra donc de savoir parmi toutes les fois où le modèle a crié au loup combien de fois avait-il raison. La precision est défini par

$$precision = \frac{VP}{VP + FP}$$

Et enfin le dernier métrique est la **surface sous la courbe de ROC (AUC)**. La courbe caractéristique de fonctionnement ou *receiver operating characteristic curve* (ROC) est une courbe où l'axe des X représente le taux de prédictions FP ($\%FP = FP/(FP + VN)$) du modèle et l'axe des Y le taux de prédictions VP ($\%VP = VP/(VP+FN)$), ce dernier est un synonyme du rappel. Le point idéal sur la courbe ROC serait donc le point (0,1), c'est-à-dire que tous les exemples positifs sont classés correctement et aucun exemple négatif n'est classé à tort comme positif. La courbe représente le taux de FP et de VP du modèle avec différents seuil de classification.

La surface sous la courbe de ROC ou *Area under the ROC curve (AUC)* est une probabilité qui indique le pourcentage de chances que le modèle de classification fasse correctement la distinction entre une instance positive choisie au hasard et une instance négative choisie au hasard. C'est une valeur entre 0 et 1, où 1 veut dire que le modèle a un pouvoir de prédiction parfait et 0.5 veut dire qu'il n'est pas meilleur qu'une supposition aléatoire.

Les modèles que j'ai entraîné tous été évalués par rapport à ces quatres métriques et les résultats sont présentés dans la section 5.2.

	Accuracy	AUC	Precision	Recall
Réseau de neuronnes	0.71494	0.76158	0.07244	0.69391
Régression logistique	0.69963	0.75756	0.06981	0.67919
Boost de gradient	0.81804	**0.83436**	0.11017	0.6608
Forêt aléatoire	**0.81872**	0.80397	0.10379	0.60972
Détection d'anomalies	0.54393	0.5705	**0.44624**	**0.7174**

Table 5.2 – Tableau comparatif des modèles

5.2 Évaluation comparative

Les quatres modèles de classification : la régression logistique (section 4.6), la forêt aléatoire (section 4.3), le boost de gradient (section 4.4), et les réseaux de neuronnes artificiels (section 4.7) ont été entraînés et ajustés sur 95% des données fournies par Home Credit Group et testés sur les 5% restants (soit à peu près 77.000 exemples).

La détection d'anomalies avec les machines à vecteur de support (section 4.8) a été entraîné sur seulement 100.000 exemples de la classe majoritaire (classe 0), faute de moyen de calcul plus rapides et testé sur un ensemble qui contient tous les exemples de la classe minoritaire (classe 1) et 5% des exemples de la classe majoritaire (soit 121.928 exemples). La table 5.2 présente les résultats des tests des cinq modèles dans la tâche de prédire les défauts de paiement sur leur ensemble de test respectifs, les valeurs de chaque métriques varient entre 0 et 1, avec 0 le plus mauvais score et 1 le meilleur.

Les meilleurs scores pour chaque métrique sont en gras. On remarque que les modèles de classification se valent globalement bien que parmi eux le Boost de gradient se démarque avec tous les métriques et conforte un peu plus la réputation qu'on lui connait pour la tâche de prédiction des défauts de paiement. Parmi les modèles de classification ceux qui ont le meilleur rappel sont le réseau de neuronnes et la régression logistique mais avec moins de 10% de précision c'est finalement assez maigre. Notez que ces résultats sont à la suite d'une validation croisé que j'ai éffectué et ne sont donc pas dépendants d'un ensemble spécifique d'entraînement ou de test.

Le modèle de détection d'anomalies quant à lui mérite une attention particulière. J'avais voulu testé cet algorithme pour une tâche de classification pour avoir un modèle qui maximise les chances de reconnaître la classe positive. Avec de tels scores de rappel et de précision, on peut considérer le test satisfaisant mais il est important de noter qu'il a été testé sur beaucoup plus de données que les modèles de classification alors qu'il a été entraîné sur beaucoup moins, de quoi relativiser les stats de justesse de classification et de l'AUC.

Troisième partie

Conclusion

Chapitre 6

Résumé et Discussion

Dans ce dernier chapitre, nous allons essayer de résumer le document et de conclure sur mes dernières ”notes” on va dire. Le but est de donner mon point de vue sur le thème, mes recherches et les futurs perpectives particulièrement dans le contexte malien.

6.1 Résumé

À travers ce mémoire, j'ai essayé d'étudier et de diagnostiquer le système de prêt moderne, ses dessous, ses origines et surtout son impact sur l'économie mondiale. Le système de prêt est aujourd'hui le support principal de l'économie moderne servant à créer des startups, financer des études, acheter des maisons etc... Il est actuellement très difficile de citer un seul domaine d'accroîssement économique qui n'en est pas presque dépendant.

Autant la tâche est plutôt simple quand le demandeur du prêt est une grande organisation, autant elle dévient délicate quand il s'agit d'individus cherchant à financer leurs projets. Accorder ou refuser un prêt à un individus est une tâche qui se repose sur une enquête traditionnellement éffectuée par les institutions prêteuses sur les démandeurs, dans ce processus l'absence d'antécédents est souvent très péjoratif pour le demandeur. Pour protéger les instititions prêteuses et les demandeurs de prêt il est nécéssaire de pouvoir attester la capacité d'un demandeur à rembourser le prêt qu'il demande.

Home Credit Group, une institution financière non bancaire internationale, a fait de ces demandeurs avec peu ou pas d'antécédents sa priorité. En février 2024, ils ont lancé une compétition sur la plateforme Kaggle où ils ont rendu publique les données de leurs enquêtes sur plus d'un million et demi de demandeurs de divers zones géographiques, parcours et antécédents qui s'étaient vu accorder un prêt, ainsi que le statut final de leurs prêts (remboursé/non remboursé) avec la

tâche de créer un modèle d'apprentissage pour distinguer les bons des mauvais prêts. Ce ensemble de données qu'ils ont publié a été le point de départ de mes recherches.

Le but des recherches était ensuite de fournir une exploration approfondie des toutes dernières méthodes d'apprentissage automatique qui ont été récemment utilisées ou qui sont actuellement étudiées pour la tâche de prédiction des défauts de paiement ; en utilisant les données fournies par Home Credit Group. En expliquant le plus clairement possible les algorithmes derrière ces différentes méthodes, les techniques et concepts associés ainsi que les implémentations que j'ai fait ou utilisés.

Pour cela j'ai implémenté, entrainé et testé 5 modèles d'apprentissage automatique dont quatre modèles de classification et un modèle de détection d'anomalies ainsi que plusieurs autre techniques de traitement des données. Les modèles entraînés ont ensuite été évalués avec des métriques comme la justesse de classification, la précision, le rappel et l'AUC.

6.2 Discussion

Dans cette dernière section je veux surtout mettre en lumières les principaux chanllenges rélatifs à mes recherches et conclure brièvement avec les futures perspectives de la recherches et quelques remarques personnelles.

Je pense qu'en y repensant, les difficultés majeures que j'ai rencontré dans mes recherches si je ne devais en citer que deux, ce serait dans l'ordre :

1. **Le traitement des données** : C'était bien la toute première fois que j'ai eu à travailler avec un ensemble de données si enorme et si complexe. Utiliser la création automatique de prédicateur était, à la base, un choix que j'ai fais pour compenser mon manque d'expérience dans cette tâche et cela s'est aussi avéver être un calvaire à gérer et surtout à optimiser. Avec un ensemble de données si grand, une machine ordinaire était juste incapable d'exécuter les différents algorithmes qui étaient liés au traitement des données et même les machines virtuelles offertes par Kaggle avaient atteint leur limites. C'était un vrai casse-tête de trouver et d'implémenter des stratégies pour essayer de gérer au mieux mes ressources. Résultat c'est la tâche qui s'est avérer la plus complexe finalement.
2. **Gérer le déséquilibre** : Pas que ça a été compliqué puisque je savais déjà à quoi m'attendre et comment y remédier, plutôt que gérer le déséquilibre des données m'a forcé à faire des choix au niveau des modèles et des métriques et surtout à faire des compromis entre les différent métriques. Ce qui au final a prolongé les temps d'entraînement et d'évaluation des modèles.

À titre personnel, l'idée d'utiliser un modèle de détection d'anomalies pour tacler les problèmes de classification bianire déséquilibré était une agréable considération dans mes recherches, je pense que bien qu'elle soit beaucoup moins éfficace en termes de temps et de ressources, c'est une méthode qui mériterait d'être plus étudiée et plus citée comme une solution pour gérer les ensembles de données déséquilibré. Je tiens toutefois à préciser que les résultats présentés dans ce document ne sont pas une limite dure des performances des différents modèles et algorithmes et qu'ils ne doivent en aucun cas être interprêté comme **l'état de l'art** de l'apprentissage automatique dans la tâche de prédiction des défauts de paiement sur les prêts.

L'application des modèles d'apprentissage automatique à la prédiction des défauts de paiement est un exemple entre autres des aspects de la technologie dans lesquels nous (en tant qu'africain et malien) sommes très en retard par rapport aux occidentaux et asiatiques. Ce qui est vraiment dommage quand on voit le potentiel de transformations de l'intelligence artficielle et de ses sous-domaines aujourd'hui et la vitesse à laquelle ils évoluent. Je pense que dans un futur proche et même plutôt très proches nous devons mettre en place des solutions pour former les gens sur comment créer et utiliser ces systèmes là au risque d'accumuler encore plus de retard dans un monde qui définitivement ne nous attendra pas.

Bibliographie

Bank, W. (2024). Domestic credit to private sector (% of GDP) [Accessed : 2024-05-29]. https://data.worldbank.org/indicator/FD.AST.PRVT.GD.ZS

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE : synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321-357.

Chollet, F., et al. (2015). Keras.

Dask Development Team. (2016). *Dask : Library for dynamic task scheduling*. http://dask.pydata.org

Diarra, Y. (2023). Covid classification [GitHub repository]. https://github.com/diarray-hub/covid_classification/tree/main/data

Diarra, Y. (2024a). anomaly-detection-for-imbalanced-classification [Kaggle Notebook]. https://www.kaggle.com/code/diarray/anomaly-detection-for-imbalanced-classification

Diarra, Y. (2024b). Automatic Feature Engineering with DFS [Kaggle Notebook]. https://www.kaggle.com/code/diarray/autofeatureengineering-dfs

Diarra, Y. (2024c). deep-feature-synthesis-home-credit-stability [Kaggle Dataset]. https://www.kaggle.com/datasets/diarray/deep-feature-synthesis-home-credit-stability

Diarra, Y. (2024d). Feature Selection [Kaggle Notebook]. https://www.kaggle.com/code/diarray/featureselection

Diarra, Y. (2024e). gradient-boosting-classifier-predict-loan-defaults [Kaggle Notebook]. https://www.kaggle.com/code/diarray/gradient-boosting-classifier-predict-loan-defaults

Diarra, Y. (2024f). Predicting loans default [GitHub repository]. https://github.com/diarray-hub/undergrad-final/tree/main/notebooks

Friedman, J. H. (2001). Greedy function approximation : a gradient boosting machine. *Annals of statistics*, 1189-1232.

GeeksForGeeks. (2023). ML | Gradient Boosting [Accessed : 2024-05-29]. https://www.geeksforgeeks.org/ml-gradient-boosting/

Gunay, D. (2023). Random Forest [Accessed : 2024-05-29]. https://medium.com/@denizgunay/random-forest-af5bde5d7e1e

Herman, D., Jelinek, T., Reade, W., Demkin, M., & Howard, A. (2024). Home Credit - Credit Risk Model Stability [Kaggle Competition dataset]. https://kaggle.com/competitions/home-credit-credit-risk-model-stability/data

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis : Towards automating data science endeavors. *2015 IEEE international conference on data science and advanced analytics (DSAA)*, 1-10.

Nielsen, M. A. (2018). Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com

Odegua, R. (2020). Predicting bank loan default with extreme gradient boosting. *arXiv preprint arXiv :2002.02011*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830.

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence : A Modern Approach (4th Edition)*. Pearson. http://aima.cs.berkeley.edu/

Schölkopf, B., Williamson, R. C., Smola, A., Shawe-Taylor, J., & Platt, J. (1999). Support vector method for novelty detection. *Advances in neural information processing systems*, *12*.